

A METHOD OF SCHEDULING EXECUTIONS OF PERIODIC AND  
ASYNCHRONOUS REAL-TIME PROCESSES HAVING HARD OR SOFT  
DEADLINES

5 FIELD OF THE INVENTION

This invention relates to the field of scheduling methods such as scheduling of processes carried out by computer or other systems, and in particular to improved methods of scheduling execution of various types of processes.

10 BACKGROUND TO THE INVENTION

In operation of a computer system, executions of certain periodically or asynchronously occurring real-time processes must be guaranteed to be completed before specified deadlines, and in addition satisfying various constraints and dependencies, such as release times, offsets, precedence relations, and exclusion relations.

15 Embedded, real-time systems with high assurance requirements often must execute many different types of processes with complex timing and other constraints. Some of the processes may be periodic and some of them may be asynchronous. Some of the processes may have hard deadlines and some of them may have soft deadlines. For some of the processes, especially the hard real-time processes, complete knowledge about their characteristics can and must be acquired before run-time. For other processes, a prior knowledge of their worst case computation time and their data requirements may not be known.

20 Some processes may have complex constraints and dependencies between them. For example, a process may need to input data that are produced by other processes. In this case, a process may not be able to start before those other processes are completed. Such constraints are referred to herein as precedence relations.

25  
30  
35

Cancel  
1-155

00336990-06199

Exclusion relations may exist between processes when some processes must prevent simultaneous access to shared resources such as data and I/O devices by other processes. For some periodic processes, they may not be  
5 able to start immediately at the beginning of their periods. In this case, those processes have release time constraints. For some periodic processes, the beginning of their first period may not start immediately from time zero, that is, the  
10 system start time. In this case, those processes have offset constraints.

Examples of such systems include plant control systems, aviation control systems, air traffic control systems, satellite control systems, communication  
15 systems, multimedia systems, on-line financial service systems, various embedded systems such as for automotive applications, etc.

Previous systems and methods related to scheduling executions of real-time processes can be  
20 broadly divided into two categories:  
(a) systems and methods in which determination of the execution schedule of all the processes is done entirely at run-time (on-line); and  
(b) systems and methods in which determination of the  
25 execution schedule of the majority of the processes is done before run-time (off-line).

The vast majority of systems and methods related to scheduling executions of real-time processes belong to the first category above. The article "Scheduling  
30 Algorithms For Multiprogramming in a Hard-Real-Time Environment", by C. L. Liu, and J. W. Layland, J. ACM,

20, 46 - 61, 1973 is the best known representative of the priority scheduling approach. It assumes that all processes are periodic, and that the major characteristics of the processes are known before run-time, that is, the worst case execution times and period are known in advance. Fixed priorities are assigned to processes according to their periods; the shorter the period, the higher the priority. At any time, the process with the highest priority among all processes ready to run, is assigned the processor.

A schedulability analysis is performed before run-time based on the known process characteristics. If certain equations are satisfied, the actual scheduling is performed during run-time, and it can be assumed that all the deadlines will be met at run-time.

The article "Priority Ceiling Protocols: An Approach to Real-Time Synchronization", by L. Sha, R. Rajkumar, and J. P. Lehoczky IEEE Trans. On Computers", 39, 1175 - 1185, 1990, makes the same assumptions as the Liu et al article, except that in addition, processes may have critical sections guarded by semaphores, and a protocol is provided for handling them. Similar to Rate Monotonic Scheduling, a schedulability analysis is performed before run-time based on the known process characteristics; if certain equations are satisfied, the actual scheduling is performed during run-time, and it can be assumed that all the deadlines will be met at run-time.

Variations on the above basic scheme have been proposed, as surveyed by C. J. Fidge in the article "Real-Time Schedulability Tests For Preemptive

Multitasking", published in Real-Time Systems, vol 14, pp. 61-93, January 1998.

Commercial real-time operating systems perform all scheduling activities at run-time, during which at  
5 each point in time, the process with the highest priority is selected for execution.

Systems and methods that perform all scheduling activities at run-time, have the following disadvantages:

10. (a) High run-time overhead due to scheduling and context switching;
- (b) Difficulty in analyzing and predicting the run-time behavior of the system;
- (c) Difficulty in handling various application  
15 constraints and process dependencies;
- (d) Low processor utilization.

High run-time overhead is partly due to the fact that embedded, real-time applications are becoming more and more complex, with an ever increasing number of  
20 processes and additional constraints and dependencies between processes. The amount of run-time resources consumed by the scheduler in order to compute the schedule, grows very rapidly as the number of processes and constraints increase. The  
25 scheduler often needs to perform many time consuming process management functions, such as suspending and activating processes, manipulating process queues, etc. In addition, since the priority scheduler does not know the schedule before run-time, it has to assume the worst  
30 case and save/restore complete contexts each time a process is preempted by another process.



Performing all the scheduling at run-time requires the use of complex run-time mechanisms in order to achieve process synchronization and prevent simultaneous access to shared resources. The run-time behavior of the scheduler can be very difficult to analyze and predict accurately.

For example, in one published study, fixed priority scheduling was implemented using priority queues, where tasks are moved between queues by a scheduler that was released at regular intervals by a timer interrupt. It was observed that because the clock interrupt handler had a priority greater than any application task, even a high priority task could suffer long delays while lower priority tasks are moved from one queue to another. Accurately predicting the scheduler overhead proved to be a very complicated task, and the estimated scheduler overhead was substantial, even though it was assumed that the system had a total of only 20 tasks, tasks did not have critical sections, and priorities are not to change. Such difficulties would be many times greater if there existed additional complex application constraints that would have to be satisfied by the synchronization mechanism at run-time.

The original schedulability analysis given in the aforementioned paper describing the PCP protocol by Sha et. al. above, assumed that all tasks are independent tasks, that there are no precedence relations, that their release times are equal to the beginning of their periods, and all periods have a common start time. It is difficult to extend the schedulability analysis for

00336990-062199

priority scheduling to take into account application constraints that frequently exist in real-time applications, such as precedence constraints, release times that are not equal to the beginning of their  
5 periods, offsets, low jitter requirements (limits on the variation in time a computed result is output to the external environment from cycle to cycle), etc. Despite considerable effort spanning more than a decade to extend

10 the original PCP protocol to handle additional constraints, not much has been accomplished to this end, as evidenced by the absence of methods that can satisfactorily deal with such constraints in the survey by Fidge mentioned above.

15 Additional application constraints increase the computational complexity of scheduling problems, which already have high computational complexity whenever processes contain critical sections. When all the scheduling is performed at run-time, the scheduler does  
20 not have enough time to find solutions for complex problems. Most systems and methods that perform all scheduling at run-time, schedule processes according to process priorities assigned by the user. However, additional application constraints are most likely to  
25 conflict with the user assigned priorities. It is not generally possible to map the many different execution orderings of processes that are required by the different application constraints in a large complex system onto a rigid hierarchy of priorities.

30 It has been suggested that higher priorities be assigned to processes with: shorter periods, higher

criticality, loIr jitter requirements, precedence constraints, etc. Consequently, the system designer is faced with the impossible task of trying to simultaneously satisfy many different application

5 constraints with one rigid hierarchy of priorities. Because of the inherent constraints built into the fixed priority scheduling model, (e.g. fixed priorities) and because scheduling is performed at run-time, attempts to take into account additional constraints

10 typically result in suggestions that either are only applicable to a few very special cases, or make drastically simplifying assumptions, which significantly reduce schedulability, or are extremely complicated, making the run-time behavior of the system very

15 difficult to analyze and predict.

In general, systems and methods that perform all scheduling activities at run-time achieve lower processor utilization than systems and methods that schedule processes before run-time. This is largely due

20 to the fact that when all scheduling activities are performed at run-time, the scheduler does not have enough time to find good solutions to complex scheduling problems, and can only afford to use crude, suboptimal methods that do not take into account all the available

25 information.

Systems that use priority scheduling methods have a much smaller chance of satisfying timing constraints, because priority-driven methods are only capable of producing a very limited subset of the

30 possible schedules for a given set of processes. This severely restricts the capability of priority-driven

systems and methods to satisfy timing and resource sharing constraints at run-time.

In general, the smaller the set of schedules that can be produced by a scheduling system or method, the smaller the chances are of finding a feasible schedule, and, the lower the level of processor utilization that can be achieved. With systems that use optimal methods that compute the schedule off-line, it is possible to achieve higher levels of resource utilization than those achievable by priority-driven systems. Hence, using priority-driven systems may increase the cost of the system to non-competitive levels.

When processes are scheduled at run-time, the scheduling strategy must avoid deadlocks. In general, deadlock avoidance at run-time requires that the run-time synchronization mechanism be conservative, resulting in situations where a process is blocked by the run-time synchronization mechanism, even though it could have proceeded without causing deadlock. This reduces further the level of processor utilization.

In contrast to conventional approaches where most of the processes are scheduled at run-time, with pre-run-time scheduling the schedule for most of the processes is computed off-line; this approach requires that the major characteristics of the processes in the system be known, or bounded, in advance. It is known that it is possible to use pre-run-time scheduling to schedule periodic processes. This consists of computing off-line a schedule for the entire set of periodic

processes occurring within a time period that is equal to the least common multiple of the periods of the given set of processes, then at run-time executing the periodic processes in accordance with the previously  
5 computed schedule.

In pre-run-time scheduling, several alternative schedules may be computed off-line for a given time period, each such schedule corresponding to a different "mode" of operation. A small run-time scheduler can be  
10 used to select among the alternative schedules in response to external or internal events. This small run-time scheduler can also be used to allocate resources for asynchronous processes that have not been converted into periodic processes.

15 It is possible to translate an asynchronous process into an equivalent periodic process, if the minimum time between two consecutive requests is known in advance, and the deadline is not too short. Thus it is also possible to schedule such asynchronous processes  
20 using pre-run-time scheduling See "The Design of Real-Time Programming Systems Based On Process Models", Proc. 1984 IEEE Real-time systems Symposium, 5-17, 1984.

Systems and methods that perform scheduling before run-time, have the following advantages:

- 25 (a) ability to effectively handle complex constraints and dependencies;  
(b) lower run-time overhead;  
(c) higher processor utilization; and  
(d) ease of predicting the system's behaviour.

30 In the majority of real-time applications, the bulk of the processing is performed by periodic

processes for which the major characteristics of the processes, including offsets, release times, worst-case execution times, deadlines, precedence and exclusion relations, and any other constraints, are known before  
5 run-time. For asynchronous processes, generally their worst-case computation times, deadlines, and the minimum time between two consecutive requests (interarrival times) are known in advance. Asynchronous processes normally are few in number, and often can be converted  
10 into new periodic processes that can be used to service the corresponding asynchronous process requests in a manner similar to polling. Thus it is not only possible, but highly desirable to schedule all the periodic processes, including the new periodic processes  
15 that are converted from some of the asynchronous processes, before run-time, rather than scheduling them at run-time.

For the processes whose characteristics are known before run-time, such as periodic processes, one  
20 may realize important advantages if the schedule is computed before run-time, instead of waiting until run-time to schedule them. This is because when scheduling is done before run-time, there is almost no limit on the running time of the scheduler, optimal scheduling  
25 methods can be used to maximize the possibility of finding a feasible schedule for the set of processes to be scheduled and to handle complex constraints and dependencies. In contrast, when processes are scheduled at run-time, the time available to the scheduler is very  
30 limited. This limits the ability of the scheduler to

find a feasible schedule and to take into account the different types of constraints and knowledge about the system processes. Once the schedule for the periodic processes has been  
5 defined before run-time, the run-time scheduler can also use this information to schedule asynchronous processes more efficiently.

Other reasons for performing scheduling before run-time include: this greatly reduces the run-time  
10 resource requirements needed for scheduling and context switching. With pre-run-time scheduling, it is possible to avoid using sophisticated run-time synchronization mechanisms by directly defining precedence relations and exclusion relations on pairs of process segments to  
15 achieve process synchronization and prevent simultaneous access to shared resources. Because the schedule is known in advance, automatic code optimization is possible; one can determine in advance the minimum amount of information that needs to be saved and  
20 restored, and one can switch processor execution from one process to another process through very simple mechanisms such as procedure calls, or simply by concatenating code when no context needs to be saved or restored, which greatly reduces the amount of run-time  
25 overhead.

When the use of sophisticated run-time synchronization mechanisms is avoided, the benefits are multi-fold: not only is the amount of run-time overhead reduced, but it is also much easier to analyze and  
30 predict the run-time behavior of the system. Compared with the complex schedulability analysis required when

run-time synchronization mechanisms are used, it is much more straightforward to verify that all processes will meet their deadlines and that all the additional application constraints will be satisfied in an off-line  
5 computed schedule.

In recent years, there has been an increased interest in systems and methods for the purpose of automating the process of pre-run-time scheduling, as described in the article by S.R. Faulk and D.L. Parnas  
10 "On Synchronization in Hard-Real-time Systems", Commun. ACM vol 31, pp.274-287, March, 1988. Cyclic executives, a form of pre-run-time scheduling, have been used in safety critical systems, e.g. as described by G.D. Carlow in the article "Architecture of the Space Shuttle  
15 Primary Avionics Software System", Commun. ACM, Sept. 1984. However, in the past, cyclic executives have mainly been constructed by hand, and are difficult to construct and maintain. In the aforementioned article by A.K. Mok, a technique for transforming an asynchronous  
20 process into an equivalent periodic process was introduced. Methods for solving the problem of scheduling processes with release times, deadlines, exclusion and precedence relations are given in the article by J.Xu and D.Parnas in the articles "Scheduling  
25 Processes with Release Times, Deadlines, Precedence and Exclusion Relations", IEEE Trans. on Software Engineering, vol 16, pp 360-369, Mar. 1990, by J.Xu and D.L. Parnas in "Pre-run-time Scheduling of Processes with Exclusion Relations on Nested or Overlapping  
30 Critical Sections", Proc. Eleventh Annual IEEE International Phoenix Conference on Computers and



Communications, IPCCC-92, Scottsdale, Arizona, April 1-3, 1992, and by J.Xu and D.L. Parnas in "On Satisfying Timing Constraints in Hard-Real-Time Systems", IEEE Trans. on Software Engineering, vol 19, pp1-17, Jan.

5 1993, which are incorporated herein by reference.

However, until now, unsolved problems have been main obstacles to fully automating the process of constructing scheduling systems that combine the pre-run-time scheduling of periodic processes with the run-  
10 time scheduling of asynchronous processes, as follows:

(1) Previously there did not exist any system or method that integrates the run-time scheduling of asynchronous processes with the pre-run-time scheduling of periodic processes, that could effectively satisfy exclusion  
15 relations, precedence relations, offsets and release times constraints between the periodic processes, as well as any exclusion relation between a periodic process and an asynchronous process, and any exclusion relation between two asynchronous processes, while  
20 making efficient use of available processor capacity, and maximizing the chances of satisfying all the timing constraints.

Previous systems and methods for scheduling periodic processes and asynchronous processes, either  
25 perform all the scheduling activities at run-time, or assume that any process can be preempted by any other process at any time (no exclusion relations can be enforced, so one cannot prevent certain data inconsistencies that are caused by more than one process  
30 simultaneously accessing shared data), or assume that all aperiodic processes have soft deadlines, or assume

- that all processes are non-preemptable, which reduces the system's flexibility in meeting deadlines, or use ad hoc schemes that do not make efficient use of available processor capacity, and reduce the chances of satisfying
- 5 all the timing constraints. Examples of such systems and methods are described in the articles "A Reservation-based Algorithm for Scheduling Both Periodic and Aperiodic Real-time Tasks", by K.G. Shin and Y.C. Chang, IEEE Trans. on Computers, vol 44, pp. 1405-1419,
- 10 Dec. 1995, "An Optimal Algorithm For Scheduling Soft Aperiodic Tasks in Dynamic-Priority Preemptive Systems", by I. Ripoli, A. Crespo and A. Garcia-Fornes, IEEE Trans. on Software Eng., vol 23, pp. 388-400, June 1997, and "The Deferrable Server Algorithm For Enhanced
- 15 Aperiodic Responsiveness in Hard Real-Time Environments", IEEE Trans. on Computers, vol 44, pp.73-91, Jan 1995.
- (2) Previously there did not exist any system or method for systematically adjusting the period lengths of periodic processes, such that the adjusted period
- 20 lengths are sufficiently close to the original period lengths, to satisfy the processor utilization level required by the application, and maximize the chances of finding a feasible schedule, while at the same time ensuring that the adjusted period lengths are as closely
- 25 harmonically related to each other (having a smaller LCM value) as possible, in order to reduce the schedule length and the number of instances of new processes, and reduce storage requirements and system overhead.
- 30 Previous systems and methods use ad hoc methods that do not make the most efficient use of available

processor capacity. For example, in cyclic executives, each periodic application task is required to complete within a fixed size frame, which is of the same size for all periodic application tasks. Such arbitrary

5 constraints seriously restrict the system's ability to meet complex timing constraints.

(3) Previously there did not exist any system or method for systematically determining which asynchronous processes should be converted into new periodic  
10 processes, and which asynchronous processes should remain asynchronous, based on whether the ratio of the processor capacity that needs to be reserved for the new periodic process, to the processor capacity that needs to be reserved for the asynchronous process if  
15 unconverted, exceeds a specified threshold.

Another embodiment of the present invention provides a system and methods for systematically adjusting the period lengths of periodic processes, such that the adjusted period lengths are sufficiently close  
20 to the original period lengths, to satisfy the processor utilization level required by the application, and maximize the chances of finding a feasible schedule, while at the same time ensuring that the adjusted period lengths are as closely harmonically related to each  
25 other (having a smaller LCM value) as possible, in order to reduce the schedule length and the number of instances of new processes, and reduce storage requirements and system overhead.

There are certain asynchronous processes that  
30 cannot be converted into periodic processes at all, or

if converted, may take up far too much processor capacity compared with allowing them to remain asynchronous. For example, an asynchronous process with a very short deadline, a short worst-case execution time, and with long interarrival times, could require that a overly high proportion, sometimes even exceeding one hundred percent, of the processor's capacity be reserved for that one single process if that process is converted into a periodic process for which it is guaranteed to meet it's deadline. But that same process, may require far less processor capacity be reserved for it, if it was not converted into a periodic process, and scheduled for execution at run-time as soon as it arrives in the system.

Previous systems and methods either indiscriminately schedule every asynchronous process at run-time as soon as it arrives in the system, or indiscriminately try to convert every asynchronous process into a periodic process; or rely on ad hoc rules of thumb.

(4) Previously there did not exist any system or method for systematically converting a given set of asynchronous processes into a set of new periodic processes that could make the most efficient use of available processor capacity, and maximize the chances of satisfying all the timing constraints.

In the aforementioned article by A.K. Mok, a technique for converting one single asynchronous process into a periodic process was described. However, the technique described in the Mok article did not consider the case of non-zero offsets, that is, non-zero

intervals between time 0, that is, the system start time, and the beginning of their first periods. If only zero offsets are allowed, the chances of satisfying all the given timing constraints is decreased considerably.

- 5 Furthermore, the described technique only deals with one process. When more than one process must be converted into periodic processes, the new periodic processes may have timing conflicts with each other and with the original set of asynchronous and periodic processes.
- 10 For example, a direct application of the above technique might result in more than one periodic process each having a release time of zero and a deadline equal to the computation time.

- Previous systems and methods use ad hoc methods
- 15 that do not make the most efficient use of available processor capacity. For example, in cyclic executives, each periodic application task is required to complete within a fixed size frame, which is of the same size for all periodic application tasks. Such arbitrary
- 20 constraints seriously restrict the system's ability to meet complex timing constraints.

#### SUMMARY OF THE INVENTION

- An embodiment of the present invention provides a system and method in which pre-run-time scheduling
- 25 techniques are combined with run-time scheduling techniques, where pre-run-time scheduling is used for scheduling the executions of periodic processes, including new periodic processes converted from a subset of the asynchronous processes, in order to satisfy
- 30 complex relations and constraints such as exclusion relations, precedence relations, and offset constraints,

release time constraints defined on the periodic processes, and any exclusion relation between a periodic process and an asynchronous process, and any exclusion relation between two asynchronous processes, and  
5 deadline constraints of all periodic processes, while run-time scheduling is used to schedule a subset of the asynchronous processes that would have consumed too much processor capacity if converted into periodic processes, so that efficient use is made of available processor  
10 capacity, and the chances of satisfying all the timing constraints of all processes is increased as much as possible.

In accordance with another embodiment, a pre-run-time scheduler may use existing methods that  
15 statically schedules a set of processes (including manual methods to satisfy any special requirements if necessary), to construct a feasible pre-run-time schedule in which processor capacity is reserved in the form of time slots in the pre-run-time schedule for each  
20 periodic process. The time slot for each periodic process also includes "room" for the execution of asynchronous processes that have less latitude than that periodic process in meeting their respective deadlines, to allow such asynchronous processes to preempt the  
25 execution of that periodic process at run-time. The pre-run-time scheduler adjusts the lengths of the periods using specified parameters which achieve a suitable balance between the utilization level and the length of the pre-run-time schedule. The pre-run-time  
30 scheduler is able to schedule periodic processes that have offsets, i.e., intervals between the

09336990-062199  
667290-06692660

start of a periodic process' first period and time zero, that is, the system's start time, and is able to take advantage of any flexibility in periodic process offsets to increase schedulability. The pre-run-time scheduler  
5 thereby is able to guarantee that every periodic process will always be able to meet its deadline, while providing good response times for asynchronous processes, without requiring any change in the methods used in any of the other parts, steps or embodiments of  
10 the present invention. The system and methods have the flexibility to employ sophisticated static scheduling methods to satisfy complex relations and constraints, such as exclusion relations, precedence relations, offsets and release times defined on the periodic  
15 processes, and also have the flexibility to incorporate and take advantage of any future new static scheduling method for satisfying any additional desired constraints among the most important and numerous type of processes in real-time applications, the periodic processes, while  
20 making efficient use of available processor capacity, and increasing as much as possible the chances of satisfying the timing constraints all the processes. Thus the present invention is able to satisfy more complex application constraints and achieve higher  
25 chances of satisfying complex application constraints on periodic processes than previous systems and methods.

In accordance with an embodiment of the invention, a run-time scheduler uses the information about the beginning times and end times of the time  
30 slots reserved for the execution of periodic processes (including those new periodic processes that were

converted from asynchronous processes) in the pre-run-time schedule, as well as the a priori knowledge of the processes characteristics, to make more informed decisions and satisfy any exclusion relation between a  
5 periodic process and an asynchronous process, and any exclusion relation between two asynchronous processes, while making efficient use of available processor capacity, and achieving higher chances of satisfying all the timing constraints of the periodic processes, when  
10 scheduling the execution of asynchronous processes. For example, an embodiment of the present invention makes it possible to completely avoid blocking of a periodic process with a shorter deadline by an asynchronous process with a longer deadline, thus achieving higher  
15 schedulability of periodic processes than previous systems and methods that schedule all processes at run-time.

In accordance with another embodiment, a run-time scheduler can use the information about the  
20 beginning times and end times of the time slots reserved for the execution of periodic processes (including those new periodic processes that were converted from asynchronous processes) in the pre-run-time schedule, as well as the a priori knowledge of the processes  
25 characteristics. A significant portion of the parameters used by the asynchronous process scheduler to make scheduling decisions are known before run-time so it is possible to pre-compute major portions of the conditions that are used for decision making. Hence the  
30 amount of computation that needs to be performed for scheduling purposes at run-time can be minimized, while



making efficient use of available processor capacity,  
and increasing as much as possible the chances of  
satisfying all the timing constraints of the  
asynchronous processes. For example, an embodiment of  
5 the present invention makes it possible to create before  
run-time, a table of "safe starting time intervals" for  
each asynchronous process, and achieve low run-time  
overhead than previous systems and methods by allowing  
the asynchronous processes to be scheduled by simple  
10 table lookup.

In accordance with another embodiment of the  
invention, bounds on the worst-case response times of  
asynchronous processes are computed, that are more  
accurate (tighter) than that achievable with previous  
15 systems and methods, by using a simulation procedure  
that takes into account the beginning times and end  
times of the time slots reserved for the execution of  
periodic processes including those new periodic  
processes that were converted from asynchronous  
20 processes) in the pre-run-time schedule, as well as the  
a priori knowledge of the processes characteristics,  
when simulating all possible worst-case scenarios of the  
executions of each asynchronous process.

In accordance with another embodiment, a system  
25 and methods can schedule the executions of both periodic  
and asynchronous real-time processes with hard or soft  
deadlines, with different a priori knowledge of the  
process characteristics, and with constraints and  
dependencies, such as offsets, release times, precedence  
30 relations, and exclusion relations, in real-time on a  
single processor. This exploits to a maximum extent any

knowledge about processes' characteristics that are available to the scheduler both before run-time and during run-time, so that processor capacity is used to satisfy the constraints

5 and dependencies of periodic and asynchronous processes with hard deadline process as a first priority. Then any remaining processor capacity is used to guarantee that processes with soft deadlines and known characteristics will also be completed before pre-  
10 determined time limits as a second priority. Then any remaining processor capacity is used to execute any asynchronous process with unknown characteristics on a "best-effort" basis.

In accordance with another embodiment, a system  
15 and methods schedule the executions of both periodic and asynchronous real-time processes with hard or soft deadlines, with different a priori knowledge of the process characteristics, and with constraints and  
20 dependencies, such as offsets, release times, precedence relations, and exclusion relations, in real-time on a single processor. This exploits to a maximum extent any knowledge about processes' characteristics that are available to the scheduler both before run-time and  
25 during run-time, in order to:

- (a) effectively handle complex application constraints and dependencies between the real-time processes;
- (b) minimize run-time overhead;
- 30 (c) make the most efficient use of available processor capacity,

(d) maximize the chances of satisfying all the timing constraints, and  
(e) provide firm and tight response time guarantees for all the processes whose characteristics are known before  
5 run-time;  
(f) make it easier to verify that all timing constraints and dependencies will always be satisfied.

Thus the present invention is able to schedule a wider variety of processes with a wider variety of  
10 constraints compared with previous systems and methods.

In accordance with another embodiment, which asynchronous processes should be converted into new periodic processes is automatically determined, and which asynchronous processes should remain asynchronous.  
15 This is based on whether the ratio of the processor capacity that needs to be reserved for the new periodic process, to the processor capacity that needs to be reserved for the asynchronous process if unconverted, exceeds a user or otherwise determined specified  
20 threshold.

An embodiment of the invention systematically converts a given set of asynchronous processes into a set of new periodic processes that could make the most efficient use of available processor capacity, and  
25 increase as much as possible the chances of satisfying all the timing constraints.

All of the above can be achieved while the process of constructing scheduling systems that combine pre-run-time scheduling with run-time scheduling of  
30 periodic and asynchronous processes is fully automated, while the most efficient use of available processor

capacity is achieved, the chances of satisfying all the timing constraints is increased as much as possible. Thus the present invention can achieve a much higher degree of automation, and substantially reduce the cost of designing the system and of making changes to the systems to meet new requirements, reduce the chances of errors, as compared with previous systems and methods that schedule processes before run-time.

09336990-062199

An embodiment of the present invention provides a system and methods for scheduling execution of both periodic and asynchronous real-time processes with hard or soft deadlines, with different a priori knowledge of the process characteristics, such that complex relations and constraints, such as exclusion relations, precedence relations, offset constraints and release time constraints defined on the hard deadline periodic processes, and any exclusion relation between a hard deadline periodic process and a hard deadline asynchronous process, and any exclusion relation between two hard deadline asynchronous processes, and deadline constraints of all hard deadline processes, can be satisfied on a single processor.

An embodiment of the present invention integrates pre-run-time scheduling with run-time scheduling to guarantee that the executions of the processes will satisfy all the specified relations and constraints. Whenever a new set of processes arrives in the system, the system schedules their executions in two phases: a pre-run-time (off-line) phase performed by a pre-run-time scheduler, and a run-time (on-line) phase performed by a run-time scheduler.

09336990-062199

In accordance with an embodiment of the invention, in each pre-run-time phase, the pre-run-time scheduler executes five steps, as follows:

In Step 1, the pre-run-time scheduler divides asynchronous processes with hard deadlines and known characteristics, referred to herein as A-h-k processes, into two subsets. One subset of asynchronous processes, referred to herein as A-h-k-p processes, are converted into new periodic processes by the pre-run-time scheduler before run-time. When the pre-run-time scheduler converts an asynchronous process into a new periodic process, it prevents possible timing conflicts with other periodic and asynchronous processes, by reserving enough "room" (time) prior to each new periodic process's deadline, to accommodate the computation times of all the periodic and asynchronous processes that have less latitude in meeting their deadlines, to allow such processes to preempt that new periodic process if possible at run-time. The processes in the other subset of asynchronous processes, referred to herein as A-h-k-a, remain asynchronous and are scheduled by the run-time scheduler at run-time. The pre-run-time scheduler reserves processor capacity for A-h-k-a processes by adding the computation time of each A-h-k-a process to the computation time of every periodic process that has a greater latitude in meeting its deadline than that A-h-k-a process, to allow each A-h-k-a process to preempt the execution of any such periodic process if possible at run-time.

Whether each asynchronous process is converted into a new periodic process or not, is determined based

on whether the ratio of the processor capacity that  
needs to be reserved for the new periodic process, to  
the processor capacity that needs to be reserved for the  
asynchronous process if unconverted, exceeds a user or  
5 otherwise specified threshold.

In Step 2, the pre-run-time scheduler determines  
the schedulability of the set of all periodic processes  
with hard deadlines and known characteristics, referred  
to herein as P-h-k processes, which also include the new  
10 periodic processes converted from A-h-k-p processes. The  
pre-run-time scheduler constructs a pre-run-time  
schedule in which one or more time slots are reserved  
for the execution of every P-h-k process, including  
every new P-h-k process converted from an A-h-k-p  
15 process. The time slots reserved for each P-h-k process  
also include time reserved for the executions of all A-  
h-k-a processes that have less latitude in meeting their  
deadlines than that P-h-k process, and which may preempt  
the execution of that P-h-k process. The pre-run-time  
20 scheduler adjusts the lengths of the periods using for  
example user or otherwise specified parameters which  
control the balance between the utilization level and  
the length of the pre-run-time schedule.

The pre-run-time scheduler is able to schedule  
25 periodic processes that have offsets, i.e., intervals  
between the start of a periodic process' first period  
and time zero. The pre-run-time scheduler takes  
advantage of any flexibility in periodic process offsets  
to increase schedulability. An embodiment of the  
30 present invention preferably allows the pre-run-time  
scheduler to use existing methods (including manual

00336990-062169  
methods) which statically schedule set of processes, to  
construct the pre-run-time schedule of periodic  
processes in Step 2 and in Step 4 (to be described  
below), without requiring any change in the methods used  
5 in any of the other steps of the present invention.  
This allows the system and methods to have the  
flexibility to incorporate and take advantage of any  
future new static scheduling method for satisfying any  
10 additionally desired constraints among the most  
important and numerous type of processes in real-time  
applications, the periodic processes.

The pre-run-time scheduler includes a function  
"adjustperiod" which uses a sorted list of reference  
periods to adjust the length of the period of each  
15 periodic process. Each reference period is equal to  $2^i$   
 $\times 3^j \times 5^k \times 7^l \times 11^f$ , ..., for some integers  $i, j, k, l, f$ ,  
... where  $0 \leq i \leq \exp_2$ ,  $0 \leq j \leq \exp_3$ ,  $0 \leq k \leq \exp_5$ ,  $0$   
 $\leq l \leq \exp_7$ ,  $0 \leq f \leq \exp_{11}$ , ...  $\exp_2, \exp_3, \exp_5, \exp_7,$   
 $\exp_{11}$ , ..., are the upperbounds on the exponents  $i, j, k,$   
20  $l, f$ , ..., that are applied to the prime numbers 2, 3, 5,  
7, 11, .... Application dependent parameters are used to  
fine tune the exponent upperbounds which control the  
balance between the utilization level and the length of  
the pre-run-time schedule.

25 In Step 3, the pre-run-time scheduler uses  
knowledge about the time slots reserved for the P-h-k  
processes in the pre-run-time schedule, to determine,  
before run-time, the worst-case response times of all A-  
h-k-a processes. The pre-run-time scheduler preferably  
30 uses one of two methods, one a formula, the other a  
simulation procedure, for determining the worst-case

response time of each A-h-k-a process. The pre-run-time scheduler verifies the schedulability of each A-h-k-a asynchronous process by checking whether its deadline is greater than or equal to its worst-case response time.

- 5 Thus, the pre-run-time scheduler provides an a priori guarantee that all periodic and asynchronous processes with hard deadlines and known characteristics will always meet their deadlines.

- In Step 4, the pre-run-time scheduler determines
- 10 the schedulability of all the periodic processes with soft deadlines and known characteristics, called P-s-k processes, under the condition that all the P-h-k processes and A-h-k-a processes that are guaranteed to be schedulable in the previous steps are
- 15 still schedulable. The pre-run-time scheduler reconstructs the pre-run-time schedule in which one or more time slots are reserved for the execution of every P-h-k process (including every new P-h-k process converted from an A-h-k-p process), and for every P-s-k
- 20 process. The time slots reserved for each P-h-k or P-s-k process also include time reserved for the executions of all A-h-k-a processes that have deadlines that are shorter than that P-h-k or P-s-k process' deadline, and which may preempt the execution of that P-h-k or P-s-k
- 25 process. The pre-run-time scheduler uses the methods in the previous step that take into account knowledge about the time slots reserved for the P-h-k and P-s-k processes in the pre-run-time schedule to determine again, before run-time,
- 30 the worst-case response times of every A-h-k-a process.



0033590-062199  
667290-065960

In Step 5, the pre-run-time scheduler preferably uses knowledge about the time slots reserved for the P-h-k and P-s-k processes in the pre-run-time schedule to determine, before run-time, the worst-case response  
5 times of asynchronous processes with soft deadlines and known characteristics, i.e., A-s-k processes.

At the end of the pre-run-time phase, a feasible pre-run-time schedule for all the periodic processes with known characteristics should be constructed, while  
10 the worst-case response times of all the asynchronous processes with known characteristics should be determined.

During the run-time phase, a run-time scheduler uses knowledge about the time slots reserved for the  
15 periodic processes in the pre-run-time schedule to schedule the executions of all the periodic and asynchronous processes, that is, the P-h-k processes (including every new P-h-k process converted from an A-h-k-p process), P-s-k processes, A-h-k-a processes, A-s-  
20 k processes, as well as asynchronous processes with soft deadlines and unknown characteristics (referred to herein as A-s-u processes, in a way that guarantees that every periodic process's execution will complete before the end of that periodic process's time slot in the pre-  
25 run-time schedule, and all the asynchronous processes with soft deadlines and known characteristics, are guaranteed to be completed within the worst-case response time pre-determined in Step 4 and Step 5 after their arrival, so that all the constraints and  
30 dependencies of all processes with known characteristics will always be satisfied. The run-time scheduler, can

use the information about the beginning times and end times of the time slots reserved for the execution of periodic processes (including those new periodic processes that were converted from asynchronous processes) in the pre-run-time schedule, as well as the a priori knowledge of the processes characteristics, to pre-compute major portions of the conditions that are used for decision making, hence the amount of computation that needs to be performed for scheduling purposes at run-time can be minimized.

For example, the present invention makes it possible to create before run-time, a table of "safe starting time intervals" for each asynchronous process, and achieve lower run-time overhead than previous systems and methods by allowing the asynchronous processes to be scheduled by simple table lookup.

A-s-u processes are scheduled at run-time on a "best-effort" basis using the remaining processor capacity.

The present invention exploits to a maximum extent any knowledge about the characteristics that are available to the system both before run-time and during run-time, in order to:

- (a) effectively handle complex application constraints and dependencies between the real-time processes;
- (b) minimize run-time overhead;
- (c) maximize the chances of being able to guarantee that all the processes with hard deadlines will always meet their deadlines;

(d) provide firm and tight response time guarantees for all the processes whose characteristics are known before run-time; and

(e) make it easier to verify that all timing constraints and dependencies will always be satisfied.

It is believed that as compared with previous systems

and methods that perform all scheduling activities at run-time, for most real-time applications, the present invention is better suited to meeting the above for the following reasons:

(1) In most real-time applications the bulk of the computation is usually performed by periodic processes for which the characteristics are known a priori. Complex constraints and dependencies are normally defined on the periodic processes. In the present invention, all the periodic processes are scheduled before run-time, there is practically no limit on the time that can be used for scheduling the periodic processes. This allows the use of better methods to handle a great variety of application constraints and dependencies, and thus can achieve higher schedulability for the most important and most numerous type of processes in real-time applications.

(2) The run-time overhead required for scheduling and context switching is much smaller than that of the prior art.

(3) The number of asynchronous processes that the run-time scheduler needs to schedule should be small,

as in most real-time applications. In most real-time applications the number of asynchronous processes with hard deadlines is usually small.

5           (4) A significant portion of asynchronous processes can be transformed into periodic processes, if desired by the user, when using the present invention. For those asynchronous processes that are not transformed  
10 into periodic processes, their interarrival times are likely to be long.

          (5) Most of the important scheduling decisions have already been determined before run-time. In particular, the relative ordering of all the periodic  
15 processes was determined before run-time when the pre-run-time schedule was computed.

          (6) A significant portion of the parameters used by the run-time scheduler to make scheduling decisions for asynchronous processes, are known before run-time,  
20 so that major portions of the conditions that are used for decision making can be pre-computed, and the amount of computation that needs to be performed for scheduling asynchronous processes at run-time can be minimized.

          (7) From the pre-run-time schedule, it becomes  
25 known in advance exactly which periodic process may preempt which other periodic process at run-time. Thus one can use this information to minimize the amount of context switching.

          (8) Once the pre-run-time schedule has been  
30 determined for all the periodic processes, the run-time scheduler can use this knowledge to achieve higher

schedulability for the small number of asynchronous processes that needs to be scheduled at run-time.

(9) The run-time scheduler can use knowledge about the pre-run-time schedule to schedule asynchronous processes more efficiently, e.g., it would be possible to completely avoid blocking of a periodic process with less latitude in meeting its deadline by an asynchronous process with greater latitude.

(10) When determining the worst-case response times of asynchronous processes, overly pessimistic assumptions need not be made, e.g., it need not be assumed that for each process, all the periodic processes with less latitude in meeting their deadlines can arrive at the same time to delay that process. Thus tighter worst-case response times for asynchronous processes can be achieved.

(11) Using the present invention, verifying that all timing constraints will always be satisfied is much easier.

(12) Using the present invention, it becomes straightforward to verify that all the timing constraints and dependencies between the periodic processes are satisfied in a pre-run-time schedule.

(13) When using the technique of pre-run-time scheduling, timing constraints and dependencies are directly "embedded" in the pre-run-time schedule, thus for the majority of the processes, the use of complicated run-time synchronization mechanisms for which it is often extremely difficult to obtain reasonable and accurate execution time bounds, can be avoided.

09336990-062199

(14) The number of asynchronous processes is reduced, and the ordering of the periodic processes is fixed in the pre-run-time schedule. This significantly reduces the complexity of verifying that the asynchronous processes will meet timing constraints.

In accordance with another embodiment of the invention, a method of scheduling executions of both periodic and asynchronous real-time processes having hard or soft deadlines, comprises automatically generating a pre-run-time schedule comprising mapping from a specified set of periodic process executions to a sequence of time slots on a time axis, each of the time slots having a beginning time and an end time, reserving each one of the time slots for execution of one of the periodic processes, a difference between the end time and the beginning time of each of the time slots being sufficiently long that execution of all of the periodic processes, including satisfaction of predetermined constraints and relations comprising at least one of release time, worst-case computation time, period, deadline, deadline nature, offset and permitted range of offset constraints, and precedence and exclusion relations and criticality levels can be completed between the beginning time and end time of respective time slots, and executing the processes in accordance with the schedule during run-time of the processor.

In accordance with another embodiment, a method for automatically adjusting lengths of periods of a predetermined set of periodic processes, comprises storing and sorting a list of reference periods, setting

the length of the period of each periodic process to the length of the largest reference period that is no larger than an original period of the periodic process to form adjusted periods, and storing the adjusted periods for  
5 subsequent use in scheduling executions of the periodic processes.

In accordance with another embodiment, a method of processing a plurality of processes, some of which are periodic and some of which are asynchronous,  
10 comprises:

(i) prior to executing the processes on a processor:

(a) determining which asynchronous processes have less flexibility in meeting their deadlines than any of the periodic processes,

15 (b) adding the execution time of each of the less flexible asynchronous processes to the execution time of each of the periodic periodic processes,

(c) scheduling each of the periodic processes into time slots,

20 (ii) and during run-time of the processor:

(d) executing the periodic processes according to the schedule, interrupting any periodic process to execute an of said less flexible asynchronous processes for which a request to execute has been received by the  
25 processor,

(e) on receiving a request to execute an asynchronous process which has greater or equal flexibility in meeting their deadlines than any of the periodic processes, scheduling the requesting  
30 asynchronous process at a time which will not conflict

with execution and completion of any of the periodic processes, and

(f) executing the scheduled latter asynchronous process at its scheduled time.

5 In accordance with another embodiment, a method of processing a plurality of processes, some of which are periodic and some of which are asynchronous, comprises:

(i) prior to executing the processes:

10 (a) determining which asynchronous processes have less flexibility in meeting their deadlines than any of the periodic processes,

(b) adding the execution time of each of the less flexible asynchronous processes to the execution  
15 time of each of the periodic processes,

(c) scheduling each of the periodic processes into time slots,

(d) converting each asynchronous process which has greater or equal flexibility in meeting their  
20 deadlines than any of the periodic processes into a new periodic process, and scheduling the new periodic process at a time which will not conflict with execution and completion of any of the other periodic processes,

(ii) and during run-time,

25 (e) executing the periodic and new periodic processes, interrupting any of the periodic and new periodic processes to process any of the less flexible asynchronous processes for which a request to execute may be received at any time.



In accordance with another embodiment, a method of scheduling execution of processes by a processor comprises:

- (a) scheduling periodic time slots for all periodic processes which time slots each include time for each of the periodic processes and time for all asynchronous processes which have less flexibility in meeting their deadlines than do the periodic processes,
- (b) determining worst case response times of all other processes,
- (c) construct a schedule which includes the periodic time slots and sufficient intervening time to process said all other processes, and
- (d) executing the processes in accordance with the schedule and as said all other processes are required to be processed from time to time.

#### BRIEF INTRODUCTION TO THE DRAWINGS

A better understanding of the invention may be obtained by reading the detailed description of the invention below, in conjunction with the following drawings, in which:

Figure 1 is a feasibility schedule for an embodiment of the invention,

Figure 2 is a timing diagram of possible run-time execution of certain asynchronous and periodic processes,

Figures 3 and 8 are feasibility schedules for other embodiments of the invention,

Figures 4, 5, 6, 7 and 9 are timing diagrams of possible run-time execution of other asynchronous and periodic processes,

Figures 10, 11 12 and 13 are example timing diagrams of possible run-time execution of various periodic processes,

Figure 14 is another example of a feasibility  
5 schedule, Figure 15, 16 and 17 are example timing diagrams of possible run-time execution of various periodic processes,

Figure 18 is another example of a feasibility schedule,

10 Figure 19 is another example of a pre-run-time schedule,

Figures 20A - 20H are timing diagrams used in the explanation of various example cases,

Figure 21 is a timing diagram of a feasible  
15 schedule on two processors,

Figures 22 and 23 are timing diagrams of pre-run-time schedulers, on two processors, and

Figures 24 and 24A are block diagrams of systems on which the present invention can be carried out.

20 DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS OF THE INVENTION

Example embodiments will now be given which illustrate operation and advantages of the present invention as compared with systems and methods that  
25 perform all scheduling activities at run-time. It should be noted that existing systems and methods that perform all scheduling activities at run-time are not able to guarantee the schedulability of the set of processes given in these examples. Some of the  
30 embodiments will be described in pseudocode, which is a

shorthand form of English understandable to persons skilled in the field of this invention.

### Periodic Processes

5 A periodic process consists of a computation that is executed repeatedly, once in each fixed period of time. A typical use of periodic processes is to read sensor data and update the current state of internal variables and outputs.

10 A periodic process  $p$  can be described by a quintuple  $(o_p, r_p, c_p, d_p, prd_p)$ , wherein  $prd_p$  is the period,  $c_p$  is the worse case computation time required by process  $p$ ,  $d_p$  is the deadline, i.e., the duration of the time interval between the beginning of a period and the time by which an execution of process  $p$  must be  
15 completed in each period,  $r_p$  is the release time, i.e., the duration of the time interval between the beginning of a period and the earliest time that an execution of process  $p$  can be started in each period, and  $o_p$  is the offset, i.e., the duration of the time interval between  
20 the beginning of the first period and time 0.

It is assumed that  $o_p, r_p, c_p, d_p, prd_p$  as well as any other parameters expressed in time have integer values. A periodic process  $p$  can have an infinite number of periodic process executions  $\{p_0, p_1, p_2, \dots\}$ ,  
25 with one process execution for each period. For the  $i$ th process execution  $p_i$  corresponding to the  $i$ th period,  $p_i$ 's release time is  $R_{pi} = o_p + r_p + prd_p \times (i-1)$ ; and  $p_i$ 's deadline is  $D_{pi} = o_p + d_p + prd_p \times (i-1)$ . The uppercase letters  $R$  and  $D$  in  $R_p$  and  $D_p$  are used herein  
30 to denote the release time and deadline respectively of a periodic process execution of some periodic process  $p$ .

Reference is made to Figure 11 and Figure 12 for examples of periodic processes. Figure 11 illustrates the periodic process  $p_B = o_{pB}, r_{pB}, c_{pB}, d_{pB}, prd_{pB}$  where  $r_{pB} = 1, d_{pB} = e, d_{pB} = 4, prd_{pB} = 12$  and  $o_{pB} = 0$ . Figure 5 12 illustrates the periodic process  $p_D = o_{pD}, r_{pD}, c_{pD}, d_{pD}, prd_{pD}$  where  $r_{pD} = 0, c_{pD} = 4, d_{pD} = 4, prd_{pD} = 12$  and  $o_{pD} = 7$ .

### Asynchronous Processes

An example of an asynchronous process is one 10 which consists of a computation that responds to internal or external events. A typical use of an asynchronous process is to respond to operator requests. Although the precise request times for executions of an asynchronous process a are not known in advance, usually 15 the minimum amount of time between two consecutive requests  $min_a$  is known in advance. An asynchronous process a can be described by a triple  $(c_a, d_a, min_a)$ .  $c_a$  is the worse case computation time required by process a.  $d_a$  is the deadline, i.e., the duration of the time 20 interval between the time when a request is made for process a and the time by which an execution of process a must be completed. An asynchronous process a can have an infinite number of asynchronous process executions  $a_0, a_1, a_2, \dots$ , with one process execution for each 25 asynchronous request. For the  $i^{th}$  asynchronous process execution  $a_i$  which corresponds to the  $i^{th}$  request, if  $a_i$ 's request (arrival) time is  $R_{ai}$ , then  $a_i$ 's deadline is  $D_{ai} = R_{ai} + d_a$ .

The uppercase letters R and D in  $R_a$  and  $D_a$  will 30 be used herein to denote the request (arrival) time and

deadline respectively of an asynchronous process  
execution of some asynchronous process a.

### Schedules

If a periodic process p or an asynchronous  
5 process a has a computation time of  $c_p$  or  $c_a$ , then it is  
assumed that that process execution  $p_i$  or  $a_i$  is composed  
of  $c_p$  or  $c_a$  process execution units. Each processor is  
associated with a processor time axis starting from 0  
and is divided into a sequence of processor time units.

10 A schedule is a mapping from a possibly infinite  
set of process execution units to a possibly infinite  
set of processor time units on one or more processor  
time axes. The number of processor time units between 0  
and the processor time unit that is mapped to by the  
15 first unit in a process execution is called the start  
time of that process execution. The number of time  
units between 0 and the time unit subsequent to the  
processor time unit mapped to by the last unit in a  
process execution is called the completion time of that  
20 process execution. A feasible schedule is a schedule in  
which the start time of every process execution is  
greater than or equal to that process execution's  
release time or request time, and its completion time is  
less than or equal to that process execution's deadline.

25 Reference is made to Figures 1-9, 14 and 18 are  
examples of feasible schedules, wherein the horizontal  
axis is the time axis, and time period segments are  
separated by vertical lines which represent release  
times and deadlines, as will be described below.

30 It should be noted that, in order to avoid use  
in this specification of an exceedingly large number of

repetitions of use of the word "executions of process" or "executions of process i", these terms have been in many places herein abbreviated to the word "process", or to simply "i". Thus whenever there is a reference to  
5 the term "process" as related to a schedule, the term "process" or "process i", or "i" when i is the name of a process should be understood as meaning "process execution" or "the execution of process I".

#### Process Segments

- 10 Each process p may consist of a finite sequence of segments  
 $p(0), p(1), \dots, p(n(p))$ , where  $p(0)$  is the first segment and  $p(n(p))$  is the last segment in process p. Given the release time  $r_p$ , deadline  $d_p$  of process p and  
15 the computation time of each segment  $p_i$  in process p, one can easily compute the release time and deadline for each segment, as described in the aforementioned 1993 article by Xu and Parnas.

- Parallel computations can be represented by  
20 several processes, with various types of relations defined between individual segments belonging to different processes, and processes can be executed concurrently; thus requiring each process to be a sequence of segments  
25 does not pose any significant restrictions on the amount of parallelism that can be expressed.

#### Precedence and Exclusion Relations

- Various types of relations such as precedence relations and exclusion relations may exist between  
30 ordered pairs of processes segments. A process segment i is said to precede another process segment j if j can

only start execution after i has completed its computation. Precedence relations may exist between process segments when some process segments require information that is produced by other process segments.

5           A process segment i is said to exclude another process segment j if no execution of j can occur between the time that i starts its computation and the time that i completes its computation. Exclusion relations may exist between process segments when some process  
10 segments must prevent simultaneous access to shared resources such as data and I/O devices by other process segments.

#### Latitude of a Process

15           The "latitude" of a process x, denoted by  $L_x$ , is a user defined measure of the latitude in meeting process x's deadline.

For exemplary purposes, in all the examples given in the description of the present invention, I will assume that  
20 for each process x,  $L_x$  is set to the following value:

--- for each periodic process  $p_i$ ,  $L_{p_i} = d_{p_i} - r_{p_i}$

--- for each asynchronous process  $a_i$ ,  $L_{a_i} = d_{a_i}$

It should be noted that  $L_x$  can be defined differently

25 according to some other criteria, depending on the application.

(For example, for each P-h-k process or P-s-k process  $p_i$ , instead of defining  $L_{p_i} = d_{p_i} - r_{p_i}$ ,  $L_{p_i} = d_{p_i}$  could be defined, or any other criteria for defining  $L_{p_i}$  could be  
30 used.

#### Main Types of Processes

The main types of processes that are considered herein are the following:

Set P-h-k: Periodic processes with hard deadlines and known characteristics. Each such process  
5 may consist of one or more segments, with precedence relations defined on them to enforce the proper ordering of segments belonging to the same process. It is assumed that the following characteristics are known for each such process segment before run-time:

10       -- period,  
          -- worst-case execution time,  
          -- release time,  
          -- deadline,  
          -- the set of data that each segment reads and  
15 writes,  
          -- any exclusion relationships with other process segments,  
          -- any precedence relationships with other periodic process segments.

20       Set A-h-k: Asynchronous processes with hard deadlines and known characteristics. It is assumed that each such process consists of a single segment and the following are known for each such process before run-time:

25       -- deadline,  
          -- worst-case execution time,  
          -- minimum time between two consecutive requests,  
          -- the set of data that the process reads and  
writes,  
30       -- any exclusion relationships with other process segments.



Set P-s-k: Periodic processes with soft deadlines and known characteristics. Each such process consists of one or more segments, with precedence relations defined on them to enforce the proper ordering  
5 of segments belonging to the same process. It is assumed that the following are known for each such process before run-time:

-- period,  
-- worst-case execution time,  
10 -- release time,  
-- deadline,  
-- the set of data that the process reads and writes,  
-- any exclusion relationships with other process  
15 segments,  
-- any precedence relationships with other periodic process segments.

Set A-s-k: Asynchronous processes with soft deadlines and known characteristics. It is assumed that  
20 each such process consists of a single segment and the following are known for each such process before run-time:

-- deadline,  
-- worst-case execution time,  
25 -- the set of data that the process reads and writes,  
-- any exclusion relationships with other process segments.

Set A-s-u: Asynchronous processes with soft  
30 deadlines and unknown characteristics. It is assumed that each such process consists of a single segment and

nothing else is known about each such process before run-time.

In the present invention, as well as in the method described in the 1993 article by Xu and Parnas referred to above, and that can be used in the present invention, It is assumed that the basic scheduling unit is a segment. The terms "segment" and "process will also be considered as having the same meaning.

#### 10 Pre-Run-Time Phase

##### Step 1: Conversion of A-h-k-p processes

In this step asynchronous processes with hard-deadlines and known characteristics are referred to as A-h-k processes. The A-h-k processes are divided into two subsets. Different methods will be used to reserve processor capacity for the execution of the processes in each of the two subsets.

The processes in one of subsets, called A-h-k-p processes, should be converted into equivalent new periodic processes with hard deadlines. The remaining A-h-k processes are called A-h-k-a processes and processor capacity should be reserved for their execution before run-time.

For each A-h-k-a process, processor capacity in each hard-deadline periodic process should be reserved by the following.

Let  $S_p$  be the original set of P-h-k processes;

Let  $S_A$  be the original set of A-h-k processes;

Let  $S_a$  be the set of A-h-k processes that have not been converted into periodic processes;

Let  $S_p$  be the set of new periodic processes that are converted from A-h-k-p processes.

For each  $p_i \in (S_p \cup S_p)$ , calculate its "adjusted computation time"  $c_{pi}$  as follows.

$$5 \quad c_{pi}' = c_{pi} + \text{adjusted capacity}(p_i)$$

The exact adjusted capacity function that is used depends on the application characteristics: for example it may depend on the number of processors that are used, and other factors.

10 In the examples, we will assume that the following simple formula is used:

Adjusted capacity( $p_i$ ) =

$$\sum_{a_j \in S_a \wedge L_{a_j} < L_{p_i}} \left\lceil \frac{d_{p_i} - r_{p_i}}{\min_{a_j}} \right\rceil c_{a_j}$$

Above, for each process  $p_i$  in  $S_p$  (the original set of P-h-k processes) or in  $S_p$  (the new periodic processes converted from A-h-k-p processes), for every possible occurrence of any A-h-k-a process  $a_j$  between  $r_{p_i}$  and  $d_{p_i}$ , if  $L_{a_j} < L_{p_i}$  then  $a_j$ 's computation time is added to  $p_i$ 's computation time.

Example 1.

Assume 4 asynchronous processes with hard deadlines and known characteristics (A-h-k processes), and 4 periodic processes with hard deadlines and known characteristics (P-h-k processes) as follows.

$$d_{\text{new}p_i} = c_{a_i} + \text{conversion room}(a_i);$$

The exact conversion room function that is used depends on the application characteristics.

In the following two examples, first assume that the following simple formula is used.

$$\text{Conversion\_room}(x_i) =$$

5

$$\begin{aligned} a_0: & c_{a_0} = 2, d_{a_0} = 2, \min_{a_0} = 1,000; \\ a_1: & c_{a_1} = 2, d_{a_1} = 7, \min_{a_1} = 1,000; \\ a_2: & c_{a_2} = 10, d_{a_2} = 239, \min_{a_2} = 1,000; \\ a_9: & c_{a_9} = 10, d_{a_9} = 259, \min_{a_9} = 1,000; \\ p_4: & r_{p_4} = 0, c_{p_4} = 26, d_{p_4} = 200, \text{prd}_{p_4} = 200, o_{p_4} = 0; \\ p_5: & r_{p_5} = 30, c_{p_5} = 16, d_{p_5} = 50, \text{prd}_{p_5} = 200, o_{p_5} = 0; \\ p_6: & r_{p_6} = 0, c_{p_6} = 26, d_{p_6} = 200, \text{prd}_{p_6} = 200, o_{p_6} = 0; \\ p_7: & r_{p_7} = 0, c_{p_7} = 16, d_{p_7} = 200, \text{prd}_{p_7} = 200, o_{p_7} = 0. \end{aligned}$$

The adjusted computation times for  $p_4, p_5, p_6, p_7$  will respectively be:

$$\begin{aligned} c_{p_4}' &= c_{p_4} + c_{a_0} + c_{a_1} = 26 + 2 + 2 = 30; \\ c_{p_5}' &= c_{p_5} + c_{a_0} + c_{a_1} = 16 + 2 + 2 = 20; \\ c_{p_6}' &= c_{p_6} + c_{a_0} + c_{a_1} = 26 + 2 + 2 = 30; \\ c_{p_7}' &= c_{p_7} + c_{a_0} + c_{a_1} = 16 + 2 + 2 = 20. \end{aligned}$$

□

One technique for converting an asynchronous process into an equivalent periodic process is described in the aforementioned 1983 article by Moka in which an asynchronous process  $(c_a, d_a, \min_a)$  can be converted into a periodic process  $(o_p, r_p, c_p, d_p, \text{prd}_p)$  which satisfies the following conditions:  $c_p = c_a, d_a \Rightarrow d_p \Rightarrow c_a, \text{prd}_p \leq \min(d_a - d_p + 1, \min_a), r_p = 0, o_p = 0$ .

However, as noted earlier, the technique described by Moka did not consider the case of non-zero offsets, that is, non-zero length intervals between time zero and the beginning of periods of the converted periodic processes.

As will be described below, the present invention extends this technique to allow non-zero offsets. In particular, when an asynchronous process  $(c_a, d_a, \min_a)$  is converted into a periodic process  $(o_p, r_p, c_p, d_p, \text{prd}_p)$ , if it satisfies the following conditions:  $c_p = c_a, d_a \Rightarrow d_p \Rightarrow c_a, \text{prd}_p \leq \min(d_a - d_p + 1, \min_a), r_p = 0$ , the converted periodic process is allowed to have offsets that range from 0 to  $\text{prd}_p - 1$ :  $o_p \leq (\text{prd}_p - 1)$ .

By using non-zero offsets, one increases considerably the chances of satisfying all the given timing constraints.

Furthermore, the technique described by Moka only deals with one process. When more than one process must be converted into periodic processes, the new periodic processes may have timing conflicts with each other and with the original set of asynchronous and

periodic processes. For example, a direct application of the Moka technique could result in more than one periodic process each having a release time of zero and a deadline equal to the computation time.

5 In order to avoid such timing conflicts, in accordance with an embodiment of the present invention a procedure is used for converting a set of asynchronous processes into periodic processes, which also takes into account the possibility  
10 of timing conflicts with other existing asynchronous and periodic processes.

First, a procedure is introduced that converts a single asynchronous process  $a_i$  into a corresponding new  
15 periodic process "newpi". When determining newpi's deadline  $d_{\text{newpi}}$ , "room" (time) is left for all the hard deadline processes that have a shorter or equal deadline than that process's deadline, as follows:

$$20 \quad \sum_{p_j \in (S_P \cup S_p) \wedge d_{p_j} \leq d_{x_i}} \left\lceil \frac{d_{x_i}}{\text{prd}_{p_j}} \right\rceil * c_{p_j} + \sum_{a_j \in S_a \wedge d_{a_j} \leq d_{x_i} \wedge i \neq j} \left\lceil \frac{d_{x_i}}{\text{min}_{a_j}} \right\rceil * c_{a_j}$$

In the above relationship, the deadline of the new periodic process  $d_{\text{newpi}}$  appears on both the left-hand  
25 side and right-hand side thereof. The value of  $d_{\text{newpi}}$  as well as all other parameters of the new periodic process can be found with the following procedure for converting a single asynchronous process  $a_i = (c_{a_i}, d_{a_i}, \text{min}_{a_i} \in S_a$  into a periodic Process  $\text{newpi} = (O_{\text{newpi}}, r_{\text{newpi}}, c_{\text{newpi}},$   
30  $d_{\text{newpi}}, \text{prd}_{\text{newpi}}) \in S_p$ :

```

5      failure:= false;
       $r_{newp_i} := 0;$ 
       $c_{newp_i} := c_{a_i};$ 
       $d_{newp_i} := c_{a_i} + conversion\_room(a_i);$ 
      deadlinefound:= false;
      while not(deadlinefound) and not(failure) do
      begin
10          $d_{previous_i} := d_{newp_i};$ 
          $d_{newp_i} = c_{a_i} + conversion\_room(previous_i);$ 
         if  $d_{previous_i} = d_{newp_i}$  then deadlinefound:= true;
         if  $(d_{a_i} - d_{newp_i} + 1) \leq min_{a_i}$ 
         then
             $prd_{newp_i} := (d_{a_i} - d_{newp_i} + 1)$ 
         else
             $prd_{newp_i} := min_{a_i};$ 
15          $prd_{newp_i} := adjustperiod(prd_{newp_i});$ 
            {perform any necessary adjustments to  $prd_{newp_i}$ }
             $d_{new_i} := d_{a_i} - prd_{newp_i} + 1;$ 
            if  $(d_{newp_i} > d_{a_i})$  or  $(prd_{newp_i} < d_{newp_i})$ 
            then failure:= true
            else if deadlinefound
            then
20                 begin
                      $S_p := S_p \cup \{newp_i\};$ 
                      $S_a := S_a - \{a_i\};$ 
                 end;
            end;
      end
25
30

```

5           If it is assumed that the earliest time that asynchronous process  $a_i$  can make a request for execution is time  $t$ , then the range of the offset  $o_{newpi}$  is  $(0, t + prd_{newpi} - 1)$ .

Reference is made to Figures 10 and 13 for  
10 examples of conversion of an asynchronous process into a periodic process.

In Figure 10, the periodic process  $newp_A = (o_{newp_A}, r_{newp_A}, C_{newp_A}, d_{newp_A}, prd_{newp_A})$  translated from the asynchronous process  $a_A = (c_{aA}, d_{aA}, min_{aA}) = (2, 7, 8)$ , where  $r_{newp_A} =$   
15  $0, C_{newp_A} = c_{aA} = 2, d_{newp_A} = c_{aA} = 2, prd_{newp_A} = \min(d_{aA} - d_{newp_A} + 1, min_{aA}) = \min(7 - 2 + 1, 8) = 6, 0 \leq o_{newp_A} = 0, \leq prd_{newp_A} - 1 = 5$ . If the offset of periodic process  $newp_A$  is set to 0, i.e.  $o_{newp_A}$ , then the periodic process executions  $newp_{A0}, newp_{A1}, newp_{A2}, newp_{A3}, newp_{A4}, newp_{A5}...$   
20 start at the times 0, 6, 12, 18, 24, 30,... respectively, and if the asynchronous request times  $R_{a0}, R_{aA1}, R_{aA2}$  are 1, 9, 17, 27, then the asynchronous process executions  $a_{A0}, a_{A1}, a_{A2}, a_{A3}$  start at the times 6, 12, 18, 30 respectively.  $a_{A0}$  executes in the time slot of  $newp_{A1}$ ,  
25  $a_{A1}$  executes in the time slot of  $newp_{A2}$ ,  $a_{A2}$  executes in the time slot of  $newp_{A3}$ , and  $a_{A3}$  executes in the time slot of  $newp_{A5}$ .

In Figure 13, the periodic process  $newp_A = (o_{newp_A}, r_{newp_A}, C_{newp_A}, d_{newp_A}, prd_{newp_A})$  translated from the asynchronous  
30 process  $a_A = (c_{aA}, d_{aA}, min_{aA}) = (2, 7, 8)$ , where  $r_{newp_A} = 0, C_{newp_A} = c_{aA} = 2, d_{newp_A} = c_{aA} = 2, prd_{newp_A} = \min(d_{aA} -$



$d_{\text{newpA}} + 1, \min_{aA}) = \min(7-2+1, 8) = 6, 0 \leq 0_{\text{newpA}} = 0, \leq$   
 $\text{prd}_{\text{newpA}} - 1 = 5$ . If the offset of periodic process  $\text{newpA}$   
 is set to 5, the periodic process executions  $\text{newpA}_0$ ,  
 $\text{newpA}_1$ ,  $\text{newpA}_2$ ,  $\text{newpA}_3$ ,  $\text{newpA}_4$ ,  $\text{newpA}_5 \dots$  start at the times  
 5, 11, 17, 23, 29, 35, ..., and if the asynchronous always  
 makes requests at the earliest possible time and at the  
 highest possible rate, the first five asynchronous  
 request times  $R_{a0}$ ,  $R_{aA1}$ ,  $R_{aA2}$ ,  $R_{aA3}$ ,  $R_{aA4}$  are 0, 8, 16, 24,  
 32, then the asynchronous process executions  $a_{A0}$ ,  $a_{A1}$ ,  
 10  $a_{A2}$ ,  $a_{A3}$  start at the times 5, 11, 17, 29, 35  
 respectively.  $A_{A0}$  executes in the time slot of  $\text{newpA}_1$ ,  
 $a_{A1}$  executes in the time slot of  $\text{newpA}_2$ ,  $a_{A2}$  executes in  
 the time slot of  $\text{newpA}_2$ , and  $a_{A3}$  executes in the time  
 slot of  $\text{newpA}_3$ ,  $a_{A4}$  executes in the time slot of  $\text{newpA}_4$ ,  
 15 and  $a_{A5}$  executes in the time slot of  $\text{newpA}_5$ , etc.

In some cases, when the lengths of periods are  
 relatively prime, the length of the LCM of the periods  
 could become inconveniently long. A function  
 "adjustperiod" can be used to adjust  $\text{prd}_{\text{newpi}}$ , whenever  
 20 the LCM became inconveniently long.

(See Example A below and Figures 15 and 16 for  
 an example of the use of the adjustperiod Procedure).  
 The adjust period function will be described below in  
 the section related to construction of a feasible pre-  
 25 run-time schedule for the P-h\_k processes.

### Example 2

Assuming that in addition to the processes in  
 30 Example 1 above, the following A-h-k process:  
 $A =: c_{a3} = 10, d_{a3} = 114, \min_{a3} = 114$ .

If the procedure above is used to convert  $a_3$  into  
a

periodic process  $\text{newp}_3$ , prior to entering the while  
loop,

$$5 \quad d_{\text{newp}_3} = C_{a_3} + C_{p_5} + C_{a_0} + C_{a_1} = 10 + 16 + 2 + 2 = 30.$$

In the first iteration of the  
while loop,  $d_{\{\text{previous}\}} = d_{\{\text{newp}_3\}} = 30$ ;  
 $d_{\{\text{newp}_3\}} = c_{\{a_3\}} + c_{\{a_0\}} + c_{\{a_1\}}$   
 $= 10 + 2 + 2 = 14.$

$$10 \quad \text{Since } d_{\text{previous}} \neq \text{newp}_3, \text{ deadlinefound} = \text{false}.$$

In the second iteration of the while loop,  
 $d_{\text{previous}} = d_{\text{newp}_3} = 14$ ;  $d_{\text{newp}_3} = C_{a_3} + C_{a_0} + C_{a_1} = 10 + 2 + 2 =$   
14.

$$15 \quad \text{Since } d_{\text{previous}} = d_{\text{newp}_3}, \text{ deadlinefound} = \text{true},$$

$$\text{prd}_{\text{newp}_3} = (d_{a_3} - d_{\text{newp}_3} + 1) = 114 - 14 + 1 = 100.$$

If we use the adjustperiod function and  
select\_exp\_upperbounds procedure described herein with  
reference to constructing a feasible pre-run-time  
schedule for the P-h-k processes to adjust the period  
20  $\text{prd}_{\text{newp}_3}$ , by using  $\text{prd}_{\text{max}} = \text{prd}_{p_4} = 200$ , and the following  
initial exponent upperbound values:  $\text{exp}_{2_{\text{init}}} = 5$ ,  $\text{exp}_{3_{\text{init}}} = 3$ ,  
 $\text{exp}_{5_{\text{init}}} = 2$ ,  $\text{exp}_{7_{\text{init}}} = 0$ , and use the value 24 for  
the parameters  $C_m$ ,  $C_{m11}$ ,  $C_{m7}$ ,  $C_{m5}$ ,  $C_{m3}$ ,  $C_{m2}$  and  $C_d$ ,  $C_{d7}$ ,  $C_{d5}$ ,  $C_{d3}$ ,  
 $C_{d2}$  and the value 0 for  $C_{e7}$ ,  $C_{e5}$ ,  $C_{e3}$ ,  $C_{e2}$  and the value 0  
25 for  $C_{e7}$ ,  $C_{e5}$ ,  $C_{e3}$ ,  $C_{e2}$ , and the values 0, 1, 2 for  $C_{e57}$ ,  
 $C_{e35}$ ,  $C_{e23}$ , respectively, then the select\_exp\_upperbounds  
procedure will produce the following values for the  
exponent upperbounds:

$$\text{exp}_2 = 5, \text{exp}_3 = 3, \text{exp}_5 = 2, \text{exp}_7 = 0, \text{exp}_{11} =$$

30 0.



$$RPC_{a_j} = \left( \sum_{p_i \in (S_P \cup S_P) \wedge L_{a_j} \leq L_{p_i}} \frac{\lceil \frac{d_{p_i} - r_{p_i}}{\min a_j} \rceil c_{a_j}}{prd_{p_i}} \right) + \frac{c_{a_j}}{\min a_j}$$

5

In general if an asynchronous process has a long minimum time between consecutive requests, then that asynchronous process is more likely to require less processor capacity to be reserved if it is treated as an A-h-k-a process compared with treating it as an A-h-k-p process.

A procedure for converting a set of asynchronous processes

15  $\{(c_{ai}, d_{ai}, \min_{ai}) \mid i = 0, 1, \dots, n\} \subset S_A$  into a set of periodic processes  $S_P = \{(o_{pi}, r_{pi}, c_{pi}, d_{pi}, prd_{pi}) \mid i = 0, 1, \dots, n\}$  follows:

$S_P :=$  original set of P-h-k processes;

$S_A :=$  original set of A-h-k processes;

$S_a := S_A$ ;

20  $S_P :=$  emptyset;

numofchanges:=0;

changes:= true;

while (changes) and (numofchanges < someupperlimit) do  
begin

25 changes:= false;

begin

for each  $a_j \in S_A$ , in the order of increasing deadlines do

30 begin

The procedure described earlier should be used to tentatively convert each A-h-k asynchronous process  $a_j \in S_A$  into a new periodic process  $newp_j \in S_P$  with the same index. Note that this conversion depends not only  
5 on  $a_j$ , but also on the entire set of new periodic processes in  $S_P$ , the original set of P-h-k processes in  $S_P$ , and the set of A-h-k-a processes in  $S_A$

```

newp_j := Tentatively_convert_into_new_periodic_process(a_j, S_P, S_P, S_A)
if (d_newp_j ≤ d_a_j) and (prd_newp_j ≥ d_newp_j)
then
begin
  c_newp_j' = c_newp_j + adjusted_capacity(newp_j);
  RPC_newp_j := c_newp_j' / prd_newp_j;
  RPC_a_j := (Σ_{p_i ∈ (S_P ∪ S_P) ∧ L_{a_j} ≤ L_{p_i}} \frac{[\frac{d_{p_i} - r_{p_i}}{\min_{a_j}}] c_{a_j}}{prd_{p_i}}) + \frac{c_{a_j}}{\min_{a_j}};
  if threshold(a_j) * RPC_newp_j ≤ RPC_a_j
  then
  begin
    if {newp_j} ∉ S_P
    {or if the version of newp_j in S_P is different
    from the newly converted version of newp_j}
    then
    begin
      changes := true;
      S_P := S_P ∪ {newp_j};
      {add new periodic process newp_j
      converted from a_j to the periodic set S_P,
      replacing any older version of newp_j in S_P}
    end;
    if {a_j} ∈ S_A then
      S_A := S_A - {a_j};
      {remove a_j from asynchronous set S_A}
    end
  else {threshold(a_j) * RPC_newp_j > RPC_a_j}
  begin
    if {a_j} ∉ S_A then
    begin

```

```

        changes:= true;
         $S_a := S_a \cup \{a_j\}$ ;
        {add  $a_j$  to asynchronous set  $S_a$ }
    end;
    if  $\{newp_j\} \in S_p$  then
         $S_p := S_p - \{newp_j\}$ ;
        {remove  $p_j$  from periodic set  $S_p$ }
    end;
end;
end;
for each  $a_j \in S_A$ , in the order of decreasing deadlines do
begin
    (repeat the same procedure that was done in the order of
    increasing deadlines for each  $a_j \in S_A$  above)
end;
if changes then
    numofchanges:= numofchanges + 1;
end;
end;

```

15

At the end of the procedure, the final set of A-h-k-a processes is  $S_a$  and the final set of new periodic processes converted from A-h-k-p processes is  $S_p$ .

Above, the range of the offset for each new  
 20 periodic process  $newp_i$ ,  $o_{newp_i}$  is  $(0, newp_i]$ . The user can decrease or increase each "threshold( $a_j$ )" value in the procedure above, to increase or decrease the likelihood that each A-h-k process  $a_j$  will be converted into a P-h-k process, depending on the application requirements.

25 Setting the threshold value of a certain asynchronous process to a high value so that it is unlikely to be converted into a new periodic process, may in some cases increase the schedulability of that process; in other cases converting an asynchronous process into a periodic  
 30 process may increase schedulability (see example 14); but increasing schedulability may not be the only

objective in a particular application. In some cases,  
one may prefer that a certain A-h-, process should be  
converted into a new periodic process whenever possible,  
to reduce scheduling overhead and/or simplify the timing  
5 analysis and make it easier to predict the system's  
behavior.

### Example 3

Assuming that the threshold value for each A-h-k  
10 process is set as follows: threshold ( $a_0$ ) = 10,  
threshold ( $a_1$ ) = 10, threshold ( $a_2$ ) = 1, threshold ( $a_3$ ) =  
2.5, threshold ( $a_9$ ) = 1, the procedure above will  
determine the following for each of the processes in  
Examples 1 and 2:

15

$a_0$ :

$d_{newp_0} = 2$ ;  $prd_{newp_0} \leq \min(d_{a_0} - d_{newp_0} + 1, \min_{a_0}) = 2 - 2 + 1 = 1 < c_{newp_0} = 2$   
 $a_0$  will NOT be converted into a new periodic process and will remain asynchronous  
and belong to the set A-h-k-a.

$a_1$ :

$d_{newp_1} = c_{a_1} + \lceil d_{newp_1} / \min_{a_0} \rceil c_{a_0} = 2 + \lceil 4/1000 \rceil 2 = 4$   
 $prd_{newp_1} = \min(d_{a_1} - d_{newp_1} + 1, \min_{a_1}) = 7 - 4 + 1 = 4$   
 $RPC_{newp_1} = c_{newp_1} / prd_{newp_1} = 4/4 = 1$   
 $RPC_{a_1} = \lceil (d_{newp_3} - r_{newp_3}) / \min_{a_1} \rceil c_{a_1} / prd_{newp_3} + \lceil (d_{p_4} - r_{p_4}) / \min_{a_1} \rceil c_{a_1} / prd_{p_4}$   
 $+ \lceil (d_{p_5} - r_{p_5}) / \min_{a_1} \rceil c_{a_1} / prd_{p_5} + \lceil (d_{p_6} - r_{p_6}) / \min_{a_1} \rceil c_{a_1} / prd_{p_6} + \lceil (d_{p_7} - r_{p_7}) / \min_{a_1} \rceil$   
 $c_{a_1} / prd_{p_7}$   
 $+ c_{a_1} / \min_{a_1} = 2/100 + 2/200 + 2/200 + 2/200 + 2/200 + 2/1000$   
 $= 0.062 < \text{threshold}(a_1) * RPC_{newp_1} = 10 * 1 = 10$

$a_1$  will NOT be converted into a periodic process, and will remain asynchronous and  
belong to the set A-h-k-a.

$a_3$ :

$d_{newp_3} = c_{a_3} + \lceil d_{newp_3} / \min_{a_0} \rceil c_{a_0} + \lceil d_{newp_3} / \min_{a_1} \rceil c_{a_1} = 10 + \lceil 30/1000 \rceil 2 + \lceil 30/1000 \rceil 2$   
 $= 14$   
 $prd_{newp_3} = \min(d_{a_3} - d_{newp_3} + 1, \min_{a_3}) = 114 - 14 + 1 = 101$

After the adjustperiod function is applied (see Example 2):

$$prd_{newp_3} = adjustperiod(101) = 100;$$

$$d_{newp_3} = d_{a_3} - prd_{newp_3} + 1 = 114 - 100 + 1 = 15;$$

$$RPC_{newp_3} = c_{newp_3}' / prd_{newp_3} = 14/100 = 0.14;$$

$$RPC_{a_3} = [(d_{p_4} - r_{p_4}) / min_{a_3}] c_{a_3} / prd_{p_4} + [(d_{p_6} - r_{p_6}) / min_{a_3}] c_{a_3} / prd_{p_6} \\ + [(d_{p_7} - r_{p_7}) / min_{a_3}] c_{a_3} / prd_{p_7} + c_{a_3} / min_{a_3} = 20/200 + 20/200 + 20/200 + 10/113 = \\ 0.388 \geq threshold(a_3) * RPC_{newp_3} = 2.5 * 0.14 = 0.35$$

$a_3$  will be converted into a new periodic process  $newp_3 = (r_{newp_3}, c_{newp_3}, d_{newp_3}, prd_{newp_3}) = (0, 10, 14, 100)$ .

$a_2$ :

$$d_{newp_2} = c_{a_2} + [d_{newp_2} / min_{a_0}] c_{a_0} + [d_{newp_2} / min_{a_1}] c_{a_1} + [d_{newp_2} / prd_{newp_3}] c_{newp_3} \\ + [d_{newp_2} / prd_{p_5}] c_{p_5} = 10 + [40/1000]2 + [40/1000]2 + [40/100]10 + [40/200]16 = \\ 40$$

$$prd_{newp_2} = min(d_{a_2} - d_{newp_2} + 1, min_{a_2}) = 239 - 40 + 1 = 200$$

$$RPC_{newp_2} = c_{newp_2}' / prd_{newp_2} = 14/200 = 0.07$$

$$RPC_{a_2} = c_{a_2} / min_{a_2} = 10/1000 = 0.01 < threshold(a_2) * RPC_{newp_2} = 1 * 0.07 = 0.07$$

$a_2$  will NOT be converted into a periodic process, and will remain asynchronous and belong to the set A-h-k-a.

$a_9$ :

$$d_{newp_9} = c_{a_9} + [d_{newp_9} / min_{a_0}] c_{a_0} + [d_{newp_9} / min_{a_1}] c_{a_1} + [d_{newp_9} / min_{a_2}] c_{a_2} \\ + [d_{newp_9} / prd_{newp_3}] c_{newp_3} + [d_{newp_9} / prd_{p_5}] c_{p_5} = 10 + [50/1000]2 + [50/1000]2 \\ + [50/1000]10 + [50/100]10 + [50/200]16 = 50$$

$$prd_{newp_9} = min(d_{a_9} - d_{newp_9} + 1, min_{a_9}) = 259 - 50 + 1 = 200$$

$$RPC_{newp_9} = c_{newp_9}' / prd_{newp_9} = 24/200 = 0.12$$

$$RPC_{a_9} = c_{a_9} / min_{a_9} = 10/1000 = 0.01 < threshold(a_9) * RPC_{newp_9} = 1 * 0.12 = 0.$$

$a_9$  will NOT be converted into a periodic process, and will remain asynchronous and belong to the set A-h-k-a.



Except for  $prd_{newp3}$  that was adjusted from 101 to 100, all other periods in this example remain unchanged by the `adjustperiod` function.

An alternative case where  $threshold(a_3)$  is set to an arbitrary large value, ensuring that A-h-k process  $a_3$  is not converted into a periodic process, is discussed below with respect to Example 13.

There exists a circular dependency relationship between the calculation of the RPC for each A-h-k-a process and the determination of the set of A-h-k-a processes. The calculation of the RPC for each A-h-k-a process depends on the original set of P-h-k processes  $S_p$ , the set of new periodic processes  $S_p$  converted from A-h-k-p periodic processes, and the set of processes that remain asynchronous  $S_a$ .

However determining which A-h-k process should be converted into a new periodic process in turn depends on the calculation of the RPC amount for the corresponding A-h-k-a process. It is for this reason that an iterative procedure is preferred to be used for this task.

Below, all periodic processes with hard-deadlines and known characteristics (including all new periodic processes in  $S_p$  that are converted from A-h-k-p processes, and the original set of P-h-k processes in  $S_p$ , will be referred to as P-h-k processes.

Step 2:

A feasible pre-run-time schedule for the P-h-k processes is constructed. In this step, the

schedulability of the set of all periodic processes with hard deadlines and known characteristics (P-h-k processes) are determined using their adjusted computation times, and a feasible pre-run-time schedule  
5 that satisfies all the specified constraints is constructed.

In some cases, when the lengths of periods are relatively prime, the length of the LCM of the periods could become inconveniently long. One may use a  
10 function "adjustperiod" to adjust the period length of each periodic process, whenever the LCM becomes inconveniently long.

Preferred function and procedures for adjusting the periods of periodic processes in order to reduce the  
15 Least Common Multiple (LCM) of the period lengths, and the pre-run-time schedule length are as follows:

```
function adjustperiod(prd);  
begin  
    j:= 1;  
    while refprd[j] ≤ prd do  
        j:= j + 1;  
    adjustperiod:= refprd[j-1];  
end;
```

```
procedure generate_refprds(exp2, exp3, exp5, exp7, exp11);  
begin  
    count:= 1;  
    for i:= 0 to exp2 do  
        begin  
            for j:= 0 to exp3 do  
                begin  
                    for k:= 0 to exp5 do  
                        begin  
                            for l:= 0 to exp7 do  
                                begin
```

```

        for f:= 0 to exp11 do
        begin
            refprd[count]:=  $2^i * 3^j * 5^k * 7^l * 11^f$ ;
            count:= count + 1;
        end;
    end;
end;
end;
end;
for i:= 1 to count - 2 do
begin
    for j:= i + 1 to count - 1 do
    begin
        if refprd[i] > refprd[j]
        then
        begin
            temp:= refprd[i];
            refprd[i]:= refprd[j];
            refprd[j]:= temp;
        end;
    end;
end;
end;

procedure select_exp_upperbounds( $prd_{max}$ ,  $exp2_{init}$ ,  $exp3_{init}$ ,  $exp3_{init}$ ,  $exp5_{init}$ ,
 $exp7_{init}$ ,  $exp11_{init}$ ,  $C_m$ ,  $C_{m11}$ ,  $C_{m7}$ ,  $C_{m5}$ ,
 $C_{m3}$ ,  $C_{m2}$ ,  $C_d$ ,  $C_{d7}$ ,  $C_{d5}$ ,  $C_{d3}$ ,  $C_{d2}$ ,
 $C_{e7}$ ,  $C_{e5}$ ,  $C_{e3}$ ,  $C_{e2}$ ,  $C_{e57}$ ,  $C_{e35}$ ,  $C_{e23}$ );

begin
     $exp2 := exp2_{init}$ ;
     $exp3 := exp3_{init}$ ;
     $exp5 := exp5_{init}$ ;
     $exp7 := exp7_{init}$ ;
     $exp11 := exp11_{init}$ ;
     $n := 2^{exp2} * 3^{exp3} * 5^{exp5} * 7^{exp7} * 11^{exp11}$ ;
    stop:= false;
    if ( $C_m * prd_{max} > n$ ) then
    begin
        while(( $C_m * prd_{max} > n$ ) and (not(stop))) do
        begin

```



```

end;
else
begin { $C_m * prd_{max} \leq n$ }
  while((( $C_d * prd_{max} < n$ ) and (not(stop)))) do
  begin
    if ((( $C_{d7} * prd_{max} \leq (n \text{ div } 7)$ ) and ( $exp7 > C_{e7}$ ) and (( $exp5 - exp7 \leq C_{e57}$ )))
    then
    begin
      n:= n div 7;
      exp7:= exp7 - 1;
    end
  else if ((( $C_{d5} * prd_{max} \leq (n \text{ div } 5)$ ) and ( $exp5 > C_{e5}$ ) and (( $exp3 - exp5 \leq C_{e35}$ )))
  then
  begin
    n:= n div 5;
    exp5:= exp5 - 1;
  end
  else if ((( $C_{d3} * prd_{max} \leq (n \text{ div } 3)$ ) and ( $exp3 > C_{e3}$ ) and (( $exp2 - exp3 \leq C_{e23}$ )))
  then
  begin
    n:= n div 3;
    exp3:= exp3 - 1;
  end
  else if ((( $C_{d2} * prd_{max} \leq (n \text{ div } 2)$ ) and ( $exp2 > C_{e2}$ )))
  then
  begin
    n:= n div 2;
    exp2:= exp2 - 1;
  end
  else
  begin
    stop:= true;
  end;
end;
end;
end;

```

The function adjustperiod uses a sorted list of reference periods "refprd" to adjust the length of the period of each periodic process. It adjusts the length  
 5 of the period of each periodic process "prd" to the largest reference period refprd(x) that is smaller or equal to prd.

The procedure generate<sub>prd</sub> creates a sorted list of reference periods "refprd", in which each reference  
 10 period refprd(x) is equal to  $2^i \cdot 3^j \cdot 5^k \cdot 7^l \cdot 11^f$ , ..., for some integers i, j, k, l, f, ...  
 where  
 $0 \leq i \leq \text{exp2}$ ,  $0 \leq j \leq \text{exp3}$ ,  $0 \leq k \leq \text{exp5}$ ,  $0 \leq l \leq \text{exp7}$ ,  $0 \leq f \leq \text{exp11}$ , ... exp2, exp3, exp5, exp7,  
 15 exp11, ..., are the upperbounds on the exponents i, j, k, l, f, ..., that are applied to the prime numbers 2, 3, 5, 7, 11, ...

In order to avoid redundancy, we will illustrate and discuss the methods using only the first five prime  
 20 numbers 2, 3, 5, 7, 11. It is possible to use more prime numbers, that is, 13, 17, 19, etc., if larger period lengths are needed. The methods shown here can be extended to include any additional prime numbers should the need arise.

25 The procedure select\_exp\_upperbounds selects the exponent upperbound values exp2, exp3, exp5, exp7, exp11, based on the maximum length of the original periods  $\text{prd}_{\text{max}}$ , and a set of initial values  $\text{exp2}_{\text{init}}$ ,  $\text{exp3}_{\text{init}}$ ,  $\text{exp5}_{\text{init}}$ ,  $\text{exp7}_{\text{init}}$ ,  $\text{exp11}_{\text{init}}$ , for exp2, exp3,  
 30 exp5, exp7, exp11, respectively. The Least Common

Multiple (LCM) of the periods will be bounded by  $2^{\text{exp2}} * 3^{\text{exp3}} * 5^{\text{exp5}} * 7^{\text{exp7}} * 11^{\text{expl1}}$ .

There is a tradeoff in adjusting periods: shorter adjusted periods result in shorter LCMs of the periods, and consequently, shorter pre-run-time schedule lengths and fewer instances of new processes; but will increase the processor utilization and waste more processor capacity. This tradeoff can be fine tuned using the exponent upperbound values  $\text{exp2}$ ,  $\text{exp3}$ ,  $\text{exp5}$ ,  $\text{exp7}$ ,  $\text{expl1}$ .

In general, larger values of the exponent upperbounds will increase the chances of the adjusted period length being close to the original schedule length, and reduce wasted processor capacity but will increase the LCM of the periods and schedule length. When the exponent upperbound associated with a larger prime number, say  $\text{expl1}$ , is increased relative to the exponent upperbounds associated with smaller prime numbers, say  $\text{exp7}$ ,  $\text{exp5}$ ,  $\text{exp3}$ ,  $\text{exp2}$ , while the exponent upperbounds associated with the smaller prime numbers are not smaller than some minimum value, e.g.,  $\min\{\text{exp7}\} = 1$ ,  $\min\{\text{exp5}\} = 2$ ,  $\min\{\text{exp3}\} = 3$ ,  $\min\{\text{exp2}\} = 5$ , respectively, it will also have the effect of reducing wasted processor capacity but increasing the LCM and schedule length.

The procedure `select_exp_upperbounds`, allows  $\text{exp2}$ ,  $\text{exp3}$ ,  $\text{exp5}$ ,  $\text{exp7}$ ,  $\text{expl1}$ , to be fine tuned according to specific application requirements. In this procedure, larger values of the parameters  $C_m$ ,  $C_{m11}$ ,  $C_{m7}$ ,  $C_{m5}$ ,  $C_{m3}$ ,  $C_{m2}$ ,  $C_d$ ,  $C_{d7}$ ,  $C_{d5}$ ,  $C_{d3}$ ,  $C_{d2}$ ; and smaller values for the parameters  $C_{e7}$ ,  $C_{e5}$ ,  $C_{e3}$ ,  $C_{e2}$ ,  $C_{e57}$ ,  $C_{e35}$ ,  $C_{e23}$ , are more





The objective of fine tuning the values of  $\text{exp2}$ ,  $\text{exp3}$ ,  $\text{exp5}$ ,  $\text{exp7}$ ,  $\text{exp11}$ , is to create a list of reference periods  $\text{refprd}(x)$ , that will have values that are sufficiently close to the original period lengths, to satisfy the processor utilization level required by the application, and maximize the chances of finding a feasible schedule, while at the same time the values should be as closely harmonically related to each other (having a smaller LCM value) as possible, in order to reduce the schedule length and the number of instances of new processes, and reduce storage requirements and system overhead.

For example, if the following conditions are satisfied, the difference between the original period length and the adjusted period length will never exceed 10% of the original period length:

- (1) the exponent upperbounds are not smaller than the following minimum values:  
 $\min\{\text{exp7}\} = 1, \min\{\text{exp5}\} = 2, \min\{\text{exp3}\} = 3, \min\{\text{exp2}\} = 5;$
- (2) the maximum period length is not greater than  $2^{(\text{exp2} - 3)} * 3^{(\text{exp3} - 1)} * 5^{(\text{exp5})} * 7^{(\text{exp7})} * 11^{(\text{exp11})}$  (the maximum period length is not greater than the maximum LCM length divided by 24).

If the exponent upperbound  $\text{exp11}$  is not smaller than the minimum value  $\min\{\text{exp11}\} = 1$ , then the maximum period length can be further increased, while still guaranteeing the same utilization level. For example, under the following conditions, the difference between the original period length and the adjusted period

length will never exceed 10% of the original period length:

(1) the exponent upperbounds are not smaller than the following minimum values:

5  $\min\{\text{exp11}\} = 1, \min\{\text{exp7}\} = 1, \min\{\text{exp5}\} = 2, \min\{\text{exp3}\} = 3, \min\{\text{exp2}\} = 5;$

(2) the maximum period length is not greater than  $2^{(\text{exp2} - 1)} * 3^{(\text{exp3} - 2)} * 5^{(\text{exp5})} * 7^{(\text{exp7})} * 11^{(\text{exp11})}$  (the maximum period length is not greater than the maximum LCM length  
10 divided by 18).

The described system and methods have the useful property that they tend to produce adjusted period lengths that are very close to the original period lengths for processes with shorter original periods,  
15 that is, those processes that are likely to constitute the largest proportion of the computation work load; while producing adjusted period lengths that are not so close to the original periods for processes whose original period lengths are large and close to the LCM  
20 of the periods, which in many applications represent the smallest work loads. In special cases where processes with periods close to the maximum period represent a significant proportion of the computation work load, one can use larger values for the parameters  $C_m, C_{m11}, C_{m7},$   
25  $C_{m5}, C_{m3}, C_{m2},$  to produce adjusted period lengths that are closer to the original schedule lengths for processes with large original period lengths.

Example A

30

Assume that it is required to schedule processes corresponding to the following tasks:

- (1) CD-Audio tasks, that need to be serviced at a rate of 2.75 KHz, that is,  $\text{prd}_{\{\text{CD-Audio}\}} = 364 \mu\text{s}$ ;
- 5 (2) ISDN Channel tasks that need to be serviced at a rate of 1.5 KHz, that is,  $\text{prd}_{\text{ISDN}} = 667 \mu\text{s}$ ;
- (3) Voice Channel tasks that need to be serviced at a rate of 1.375 KHz,  
that is,  $\text{prd}_{\text{Voice}} = 727 \mu\text{s}$ ;
- 10 (4) Keyboard/Mouse tasks that need to be serviced periodically with periods of length  $\text{prd}_{\{\text{Keyboard/Mouse}\}} = 100 \text{ ms} = 100,000 \mu\text{s}$ .

Assume that the following initial exponent upperbound values are used:

- 15  $\text{exp2}_{\text{init}} = 5$ ,  $\text{exp3}_{\text{init}} = 3$ ,  $\text{exp5}_{\text{init}} = 2$ ,  $\text{exp7}_{\text{init}} = 1$ ,  
 $\text{expl1}_{\text{init}} = 0$ , and use the value 24 for the parameters  $C_m$ ,  $C_{m11}$ ,  $C_{m7}$ ,  $C_{m5}$ ,  $C_{m3}$ ,  $C_{m2}$ ,  
and  $C_d$ ,  $C_{d7}$ ,  $C_{d5}$ ,  $C_{d3}$ ,  $C_{d2}$ , the value 0 for  $C_{e7}$ ,  $C_{e5}$ ,  $C_{e3}$ ,  
 $C_{e2}$  and the values 0, 1, 2, for  $C_{e57}$ ,  $C_{e35}$ ,  $C_{e23}$ ,
- 20 respectively.

The `select_exp_upperbounds` procedure will produce the following values for the exponent upperbounds:  $\text{exp2} = 5$ ,  $\text{exp3} = 3$ ,  $\text{exp5} = 2$ ,  $\text{exp7} = 1$ ,  $\text{expl1} = 1$ .

- 25 After the `generate_refprd` procedure has used the above exponent upperbounds to compute the sorted list of reference periods in `refprd`, the `adjustperiod` function will use the sorted list of reference periods to compute the following adjusted periods:

- $\text{prd}_{\{\text{CD-Audio}\}} = 360$  (original 364),
- 30  $\text{prd}_{\text{ISDN}} = 660$  (original 667),
- $\text{prd}_{\text{Voice}} = 720$  (original 727),

$\text{prd}_{\{\text{Keyboard/Mouse}\}} = 92400$  (original 100000).

The LCM of the periods will be 277200, which is 3 times the length of the maximum adjusted process period length.

5        Assume now that such a high precision is not required when adjusting the periods of the processes with maximum process length, then a smaller value, say 1, may be used for the parameters  $C_m$ ,  $C_{m11}$ ,  $C_{m7}$ ,  $C_{m5}$ ,  $C_{m3}$ ,  $C_{m2}$ , and  $C_d$ ,  $C_{d7}$ ,  $C_{d5}$ ,  $C_{d3}$ ,  $C_{d2}$ , and still use the value 0  
10    for  $C_{e7}$ ,  $C_{e5}$ ,  $C_{e3}$ ,  $C_{e2}$  and the values 0, 1, 2, for  $C_{e57}$ ,  $C_{e35}$ ,  $C_{e23}$ , respectively.

Assume that the same initial exponent upperbound values are used:

$\text{exp2}_{\text{init}} = 5$ ,  $\text{exp3}_{\text{init}} = 3$ ,  $\text{exp5}_{\text{init}} = 2$ ,  
15     $\text{exp7}_{\text{init}} = 1$ ,  $\text{exp11}_{\text{init}} = 0$ . The `select_exp_upperbounds` procedure will produce the following values for the exponent upperbounds:  
 $\text{exp2} = 5$ ,  $\text{exp3} = 3$ ,  $\text{exp5} = 2$ ,  $\text{exp7} = 1$ ,  $\text{exp11} = 0$ .

After the `generate_refprd` procedure has used the  
20    above exponent upperbounds to compute the sorted list of reference periods in `refprd`, the `adjustperiod` function will use the sorted list of reference periods to compute the following adjusted periods:

$\text{Prd}_{\{\text{CD-Audio}\}} = 360$  (original 364),  
25     $\text{prd}_{\{\text{ISDN}\}} = 630$  original 667,  
       $\text{prd}_{\{\text{Voice}\}} = 720$  (original 727),  
       $\text{prd}_{\{\text{Keyboard/Mouse}\}} = 75600$  (original 100000).

The LCM of the periods will be 75600, which is 1  
times the length of the maximum adjusted process period  
30    length.

Note that in the second case, a smaller value for the parameters  $C_m$ ,  $C_{m11}$ ,  $C_{m7}$ ,  $C_{m5}$ ,  $C_{m3}$ ,  $C_{m2}$ , and  $C_d$ ,  $C_{d7}$ ,  $C_{d5}$ ,  $C_{d3}$ ,  $C_{d2}$ , resulted in smaller values for the exponent upperbounds ( $exp_{11}$  was reduced from 1 to 0), and consequently, resulted in shorter adjusted periods, where the difference is greatest for the processes with the largest periods ( $prd_{\text{(Keyboard/Mouse)}}$  was reduced from 92400 to 75600), and the LCM was also noticeably reduced from 3 times the maximum adjusted period length to 1 times the maximum adjusted period length.

The following description we includes a procedure for constructing a feasible pre-run-time schedule for a given set of periodic processes with offsets.

Procedure for constructing a feasible pre-run-time schedule for a given set of periodic processes with offsets

$S_p = [(o_{pi}, r_{pi}), c_{pi}, d_{pi}, prd_{pi}) \mid i = 0, 1, \dots, n]$ :

```

begin {procedure}
success:= false;
for all  $p_i$  let  $prd_{p_i} := \text{adjustperiod}(prd_{p_i})$ ;
 $prd_{LCM} := \text{LCM}(prd_{p_0}, prd_{p_1}, \dots, prd_{p_n})$ ;
begin {for loops}
for  $o_{p_0} := \text{offsetlowerbound}(p_0)$  to  $\text{offsetupperbound}(p_0)$  do
  for  $o_{p_1} := \text{offsetlowerbound}(p_1)$  to  $\text{offsetupperbound}(p_1)$  do
    ...
    for  $o_{p_i} := \text{offsetlowerbound}(p_i)$  to  $\text{offsetupperbound}(p_i)$  do
      ...

```

```

for  $o_{p_n} := \text{offsetlowerbound}(p_n)$  to  $\text{offsetupperbound}(p_n)$  do
begin {loop body}
    Construct a schedule for all occurrences of all the processes in  $S_p$ 
    within the interval  $[0, \max\{o_{p_i} | \forall p_i\} + 3 * \text{prd}_{LCM}]$ ;
    If the following conditions are satisfied, then set  $\text{success} := \text{true}$ ; and
    exit from the procedure:
    There must exist a point  $t$  in the schedule, such that:
    (1) The subschedule in the interval  $[t, t + \text{prd}_{LCM}]$  is equal to the
    subschedule in the interval  $[t + \text{prd}_{LCM}, t + 2 * \text{prd}_{LCM}]$ ;
    (2) All the occurrences of all processes in  $S_p$  within the interval
     $[t, t + \text{prd}_{LCM}]$  must be included in the subschedule in  $[t, t + \text{prd}_{LCM}]$ ;
    (3) All the occurrences of all processes in  $S_p$  within the interval
     $[0, t + \text{prd}_{LCM}]$  must satisfy all their respective timing constraints.

```

```

end; {loop body}
end; {for loops}
if  $\text{success}$  is true then set the "initial part of the pre-run-time schedule"  $S_0(t)$  to be
the subschedule in the interval  $[0, t]$ , and set the "repeating part of the pre-run-time
schedule"  $S_{LCM}(\text{prd}_{LCM})$  to be the subschedule in the interval  $[t, t + \text{prd}_{LCM}]$ ;
end. {procedure}

```

The optimal scheduling method in the aforementioned  
 1990 article by Xu and Parnas that constructs a feasible  
 20 schedule for a set of processes with release times,  
 deadlines, precedence and exclusion relations, can be  
 used for this purpose. If the processes are to be  
 scheduled on more than one processor, the optimal  
 scheduling method as described in the aforementioned 1993  
 25 article by Xu and Parnas that constructs a non-  
 preemptive feasible schedule for a set of processes  
 with release times, deadlines, precedence and exclusion  
 relations on multiple processors, can be used for this  
 purpose. Note that it is not strictly necessary to use  
 30 the scheduling method of the 1990 or the 1993 article.  
 One may use any method for this step, the only

requirements being that the method should be capable of constructing a feasible schedule for a set of periodic processes, in which all the specified constraints and dependencies between the processes are satisfied.

5

#### Example 4

Assume that the following precedence relations and

10 exclusion relations must be satisfied for the processes in

Examples 1-3 above:

$p_6$  precedes  $p_7$ ;

$a_2$  excludes  $a_3, a_9$ ;

15  $a_3$  excludes  $a_2, p_4, p_6$ ;

$a_9$  excludes  $a_2$ ;

$p_4$  excludes  $a_3, p_5, p_6$ ;

$p_5, p_6$  excludes  $p_4$ ;

$p_5$  excludes  $p_6$ ;

20  $p_6$  excludes  $a_3, p_5$ .

If the method in the aforementioned 1990 Xu and Parnas article is used to schedule all the P-h-k processes  $newp_3, p_4, p_5, p_6, p_7$  using their adjusted computation times, the feasible schedule will be found  
25 in Figure 1. This guarantees that all the P-h-k processes are schedulable.

In Example 4 above, a feasible schedule for the case in which the offsets of all the periodic processes are zero was shown. Below is another example to show  
30 how the procedure given above can use existing methods to schedule a set of periodic processes together with a

set of new periodic processes that are converted from asynchronous processes, where the original periodic processes and the new periodic processes may have non-zero offsets.

5

#### Example B

Assume the following A-h-k process  $a_A$  (see Fig. 10):

10  $a_A: c_{aA} = 2; d_{aA} = 7; \min_{aA} = 8.$

Assume also the following two periodic processes  $p_B$  and  $p_C$  (see Fig. 11 and Fig. 12):

$p_B: o_{pB} = 0; r_{pB} = 1; c_{pB} = 3; d_{pB} = 4$   
 $\text{prd}_{pB} = 12.$

15  $p_C: o_{pC} = 7; r_{pC} = 0; c_{pC} = 4; d_{pC} = 4$   
 $\text{prd}_{pC} = 12.$

Assume that the application requires  $p_B, p_C$  to have fixed offset values of 0 and 7 respectively.

Assuming that asynchronous process  $a_A$  may make  
20 its first request at time 0, the procedure for converting a set of asynchronous processes into a set of periodic processes given earlier, could convert  $a_A$  into the following new periodic process:

$\text{newp}_A: r_{\text{newp}_A} = 0; c_{\text{newp}_A} = 2;$   
25  $d_{\text{newp}_A} = 2; \text{prd}_{\text{newp}_A} = 6; \text{ and } 0 \leq o_{\text{newp}_A} \leq \text{prd}_{\text{newp}_A} - 1 = 5.$

The lowerbound of the offset of  $\text{newp}_A$  is 0, and the upperbound of the offset of  $\text{newp}_A$  is 5 (see Figure 10 and Figure 13).

The procedure for constructing a feasible pre-  
30 run-time schedule for a given set of periodic processes with offsets given above will try each offset value of



$o_{newpA}$  within the range of  $0, prd_{newpA} - 1 = (0, 5)$ , when trying to find a feasible schedule for  $newp_A$ ,  $p_B$ , and  $p_C$ . When the last value in that range,  $o_{newpA} = 5$  is used, the method in the 1990 article by Xu and Parnas would  
 5 find the feasible schedule illustrated in Figure 14. A feasible schedule is found for  $newp_A$ ,  $p_B$  and  $p_C$ , when  $o_{newpA} = 5$  and  $o_{pD} = 2$ . It is assumed that the application requires that  $o_{pC} = 7$ ,  $o_{pB} = 0$  and since asynchronous process  $a_A$  may make its first request at  
 10 time  $0, 0 \leq o_{newpA} \leq prd_{newpA} - 1 = 5$ . The feasible schedule consists of a non-repeating initial schedule  $S_{01}$  for the interval  $(0, 1)$ , and a repeating schedule  $S_{LCM12}$  that is of length equal to the Least Common Multiple of the periods of  $newp_A$ ,  $p_B$ ,  $p_C$ , and  $p_D$ , which  
 15 is  $LCM(6, 12, 12, 12) = 12$ .

Assume that in addition to the periodic processes  $newp_A$ ,  $p_B$ ,  $p_C$ , the following periodic process  $p_D$  exists (as shown in Figure 15):

$p_D: r_{pD} = 2, c_{pD} = 1, d_{pD} = 4,$   
 20  $prd_{pD} = 13, 0 \leq o_{pD} \leq 4.$

If the "adjustperiod" function is applied to the periods of  $newp_A$ ,  $p_B$ ,  $p_C$ , and  $p_D$ , then  $p_D$ 's period  $prd_{pD}$  will be shortened from length 13 to length 12, resulting in the modified periodic process  $p_D = o_{pD}, r_{pD}, c_{pD}, d_{pD},$   
 25  $prd_{pD}$  where  $r_{pD} = 2, c_{pD} = 1, d_{pD} = 4, prd_{pD} = 12, 0 \leq o_{pD} \leq 4$ , as shown in Figure 16.

The other periods  $prd_{newpA}$ ,  $prd_{pB}$ , and  $prd_{pC}$  remain unchanged after the adjustperiod procedure is applied. Assuming that the application allows the offset of  $p_D$  to  
 30 be in the range of  $(0, 4)$ , the procedure for constructing a feasible pre-run-time schedule for a

given set of periodic processes with offsets given above will try each offset value of  $O_{\text{newpA}}$  within the range of  $(0, \text{prd}_{\text{newpA}}-1) = (0, 5)$ , and try each offset value of  $O_{\text{pD}}$  within the specified permitted range of  $(0, 4)$ , when  
 5 trying to find a feasible schedule for  $\text{newpA}$ ,  $p_B$ ,  $p_C$ , and  $p_D$ . When the offset values  $O_{\text{newpA}} = 5$ , and  $O_{\text{pD}} = 2$ , as shown in Figure 17 are used, the method described in the 1990 Xu and Parnas article would find the feasible schedule illustrated in Figure 18.

10 As shown in Figure 17, the periodic process  $p_D = O_{\text{pD}}, r_{\text{pD}}, c_{\text{pD}}, d_{\text{pD}}, \text{prd}_{\text{pD}}$  where  $r_{\text{pD}} = 2, c_{\text{pD}} = 1, d_{\text{pD}} = 4, \text{prd}_{\text{pD}} = 12, 0 \leq O_{\text{pD}} \leq 4$ , where the offset of  $p_D$  is set to a fixed value  $O_{\text{pD}} = 2$  during run-time scheduling when it is attempted to construct a feasible pre-run-time  
 15 schedule for  $\text{newpA}$ ,  $p_B$ ,  $p_C$  and  $p_D$ .

As shown in Figure 18, a feasible schedule is shown for  $\text{newpA}$ ,  $p_B$ ,  $p_C$  and  $p_D$ , when  $O_{\text{newpA}} = 5$  and  $O_{\text{pD}} = 2$ . It is assumed that the application requires that  $O_{\text{pC}} = 7, O_{\text{pB}} = 0$ , and  $0 \leq O_{\text{pD}} \leq 4$ , and asynchronous process  
 20  $a_A$  may make its first request at time 0. The feasible schedule consists of a non-repeating initial schedule  $S_0(1)$  for the interval  $(0,1)$ , and a repeating schedule  $S_{\text{LCM}}(12)$  that is of length equal to the least common multiple of the periods of  $\text{newpA}$ ,  $p_B$ ,  $p_C$  and  $p_D$ , which is  
 25  $\text{LCM}(6, 12, 12, 12) = 12$ .

The feasible schedule consists of a non-repeating initial schedule  $S_{0(1)}$  for the interval  $[0,1]$ , and a repeating schedule  $S_{\text{LCM}(12)}$  that is of length equal to the Least Common Multiple of the periods of  $\text{newpA}$ ,  
 30  $p_B$ ,  $p_C$ , and  $p_D$ , which is  $\text{LCM}(6, 12, 12, 12) = 12$ .

- The following notation is used below to denote the beginning and end of the time slot of each process in a pre-run-time schedule, the actual start time and actual completion time of each process, and the most recent actual arrival time of an asynchronous process at run-time. This notation will be used extensively below.
- $s(p)$  is the time of the beginning of the time slot that was reserved for periodic process  $p$  in the pre-run-time schedule.
- $S'(x)$  is the actual time that periodic process or asynchronous process  $x$  was/will be put into execution at run-time. At any time  $t$ , if periodic or asynchronous process  $x$  has been put into execution after or at time  $t$ , then  $t \leq s'(x)$  is true, otherwise  $\neg t \leq s'(x)$  is true.
- $s'(p)$  depends on the arrival times of asynchronous processes  $a_j$  and whether and at what time they preempt periodic processes.  $s'(p)$  also depends on the actual execution times of other processes that are executed prior to  $p$ 's execution at run-time.
- $e(p)$  is the time of the end of the time slot that was reserved for periodic process  $p$  in the pre-run-time schedule.
- $e'(x)$ : the actual time at which asynchronous or periodic process  $x$ 's execution ends at run-time. At any time  $t$ , if periodic or asynchronous process  $x$ 's execution has ended after or at time  $t$ , then  $t \leq e'(x)$  is true; otherwise if  $x$ 's execution has not ended before or at time  $t$ , then  $\neg t \leq e'(x)$  is true.
- $R'(a)$ : the most recent actual arrival time of asynchronous process  $a$  at run-time. At any time  $t$ , if

asynchronous process  $a$  has arrived before or at time  $t$ , then  $R'(a) \leq t$  is true, otherwise  $\neg R'(a) \leq t$  is true. At any time  $t$ , if asynchronous process  $a$  has arrived at least once before time  $t$  and after or at time 0, then 0  $\leq R'(a)$  is true, otherwise if  $a$  has never arrived before or at time  $t$ , then  $\neg 0 \leq R'(a)$  is true.

#### Example 5

10 In Figure 1, the time slot that is assigned to the P-h-k process  $p_6$  in the feasible pre-run-time schedule begins at time 114, and ends at time 144, so  $s(p_6) = 114$ ,  $e(p_6) = 144$ . The length of the time slot assigned 15 to  $p_6$  in the pre-run-time schedule is equal to the adjusted computation time of  $p_6$ , i.e.,  $c_{p_6}'$ , which includes the time reserved in case  $p_6$  is preempted by A-h-k-a processes with less latitude in an actual execution.

20 Fig. 2 shows a possible actual execution of the P-h-k processes when they are scheduled at run-time together with the A-h-k-a processes  $a_0, a_1, a_2, a_9$ , together with the P-h-k processes of Figure 1. In this execution, the worst case response time of  $a_2$  of the A-h-k process  $Re_{a_2} = \max\{RE(a_2, t_s)\} = RE(a_2, 82) = e'(a_2) - R_{a_2} = 166 - 82 = 84$ . (the details are explained in 25 Example 7 below, see particularly step 3).

Note that in Figure 2 the relative order in which P-h-k processes are executed at run-time, is kept 30 the same as the relative order of the time slots that are assigned to those periodic processes in the feasible

pre-run-time schedule in Fig. 1. Note also that the length of the computation time indicated for each P-h-k process is its original computation time, not its adjusted computation time. For example, in Fig. 2,

5  $s'(p_6) = 114$ ,  $e'(p_6) = 140$ .

Step 3: Determine the worst case response times of the A-h-k-a processes

- 10 A-h-k-a processes are scheduled at run-time by the A-h-k-a Scheduler Process (see the description later which describes scheduling A-h-k-a processes), but their worst-case response times are determined before run-time in this step. Verification that each A-h-k-a process  $a_i$
- 15 is schedulable is performed by checking the condition that  $a_i$ 's latitude must be greater than or equal to its response time.

Two different methods of determining the response time of an A-h-k-a process will now be

20 described. The first method uses a formula to calculate the response time. The second method uses simulation to determine the response time. The second method gives tighter response times that can guarantee the schedulability of a larger number of cases, but requires

25 a longer computation time. The first method, while not as accurate as the second method, is faster. In practice time can be saved by applying the first method first, and only applying the second method if the response time of some A-h-k-a process determined by the

30 first method exceeds the latitude of that process.

"P-g" is the set of guaranteed periodic processes. In this step, P-g includes all the P-h-k processes, including those A-h-k-p processes that are translated into periodic processes. Later on, in Step 4, P-g is  
5 expanded to include all P-h-k and P-s-k processes.

The worst case response time of an A-h-k-a process  $a_i$  can be determined in accordance with the following method:

For all  $a_i \in A\text{-h-k-a}$ :

$$RE_{a_i} = c_{a_i} + \text{DelayA}(a_i, RE_{a_i}) + \text{DelayP}(a_i, RE_{a_i}) + B(a_i) + \text{GT}(a_i, RE_{a_i})$$

where

$$\text{DelayA}(a_i, RE_{a_i}) = \sum_{a_j \in A\text{-h-k-a} \wedge L_{a_j} \leq L_{a_i} \wedge j \neq i} \left\lceil \frac{RE_{a_i}}{\text{min}_{a_j}} \right\rceil \cdot c_{a_j}$$

and

$$\text{DelayP}(a_i, RE_{a_i}) = \sum_{p_j \in P\text{-g} \wedge L_{p_j} \leq L_{a_i}} \left\lceil \frac{RE_{a_i}}{\text{prd}_{p_j}} \right\rceil \cdot c_{p_j}$$

and

$$B(a_i) = \max \{ c_{a_j}, e(p_l) - s(p_l) \mid (a_j \in A\text{-h-k-a} \wedge L_{a_j} > L_{a_i} \wedge \exists x_k, x_k \in A\text{-h-k-a} \forall x_k \in P\text{-g}: a_j \text{ excludes } x_k \wedge L_{x_k} \leq L_{a_i}) \vee (p_l \in P\text{-g} \wedge L_{p_l} > L_{a_i} \wedge \exists x_k, x_k \in A\text{-h-k-a} \forall x_k \in P\text{-g}: p_l \text{ excludes } x_k \wedge L_{x_k} \leq L_{a_i}) \}$$

and

$$\text{GT}(a_i, RE_{a_i}) = \sum_{p_l \in SG1(a_i)} \left\lceil \frac{RE_{a_i}}{\text{prd}_{p_l}} \right\rceil \cdot c_{a_i} + \sum_{p_l \in SG2(a_i)} \left\lceil \frac{RE_{a_i}}{\text{prd}_{p_l}} \right\rceil \cdot c_{a_i}$$

where  $SG1(a_i) = \{p_l \mid p_l \in P\text{-g} \wedge L_{p_l} \leq L_{a_i} \wedge (a_i \text{ excludes } p_l)\}$ ;  
and  $SG2(a_i) = \{p_l \mid p_l \in P\text{-g} \wedge L_{p_l} \leq L_{a_i} \wedge (\exists a_j \in A\text{-h-k-a}: a_i \text{ excludes } a_j \wedge L_{a_j} < L_{p_l})\}$ ;

It should be noted that in the above method, the value of  $c_{pj}$  is the original computation time of  $p_j$  (it does not include the time reserved for A-h-k-a processes with less latitude).

In a first assumption, for each A-h-k-a process  $a_i$ , for each  $RE_{ai}$  computed by the above method, if  $RE_{ai} \leq L_{ai}$ ,  $a_i$  will always meet its deadline at run-time.

The following procedure can be used to compute the worst case response time of each A-h-k-a process:

```

i:= 0;
failure= false;
while i ≤ number-of-A-h-k-a-processes and not(failure) do
begin
    if  $a_i \in$  A-h-k-a
    then
    begin
         $RE_{new_i} := c_{a_i}$ ;
        responsetimefound:= false;
        while not(responsetimefound) and not(failure) do
        begin
             $RE_{previous_i} := RE_{new_i}$ ;
             $RE_{new_i} = \text{DelayA}(a_i, RE_{previous_i}) + \text{DelayP}(a_i, RE_{previous_i})$ 
                 $+ B(a_i) + GT(a_i, RE_{previous_i})$ ;
            if  $RE_{previous_i} = RE_{new_i}$ 
            then
            begin
                 $RE_{a_i} := RE_{new_i}$ ;
                responsetimefound:= true;
            end
            if ( $RE_{new_i} > L_{a_i}$ )
            then failure:= true
        end;
    end;
    i:= i + 1;
end
end

```

# Example 6

If the procedure and formula described in this specification is used to calculate the response times of the A-h-k-a processes in Examples 1-5 above, the following should be computed:

$$\begin{aligned}
 RE_{a_0} &= c_{a_0} = 2 \leq L_{a_0} = 2 \\
 RE_{a_1} &= c_{a_1} + [RE_{a_1}/min_{a_0}]c_{a_0} = 2 + [7/1000]2 = 4 \leq L_{a_1} = 7 \\
 DelayA(a_2, RE_{a_2}) &= [RE_{a_2}/min_{a_0}]c_{a_0} + [RE_{a_2}/min_{a_1}]c_{a_1} = 2 + 2 = 4 \\
 DelayP(a_2, RE_{a_2}) &= [RE_{a_2}/prd_{newp_3}]c_{newp_3} + [RE_{a_2}/prd_{p_4}]c_{p_4} + [RE_{a_2}/prd_{p_5}]c_{p_5} \\
 &+ [RE_{a_2}/prd_{p_6}]c_{p_6} + [RE_{a_2}/prd_{p_7}]c_{p_7} = 20 + 26 + 16 + 26 + 16 = 104 \\
 B(a_2) &= c_{a_2} = 10 \\
 GT(a_2, RE_{a_2}) &= [RE_{a_2}/prd_{newp_3}]c_{a_2} = 20 \\
 RE_{a_2} &= c_{a_2} + DelayA(a_2, RE_{a_2}) + DelayP(a_2, RE_{a_2}) + B(a_2) + GT(a_2, RE_{a_2}) = 10 + \\
 &4 + 104 + 10 + 20 = 148 \leq L_{a_2} = 239 \\
 DelayA(a_9, RE_{a_9}) &= [RE_{a_9}/min_{a_0}]c_{a_0} + [RE_{a_9}/min_{a_1}]c_{a_1} + [RE_{a_9}/min_{a_9}]c_{a_9} = 2 + 2 \\
 &+ 10 = 14 \\
 DelayP(a_9, RE_{a_9}) &= [RE_{a_9}/prd_{newp_3}]c_{newp_3} + [RE_{a_9}/prd_{p_4}]c_{p_4} + [RE_{a_9}/prd_{p_5}]c_{p_5} \\
 &+ [RE_{a_9}/prd_{p_6}]c_{p_6} + [RE_{a_9}/prd_{p_7}]c_{p_7} = 20 + 26 + 16 + 26 + 16 = 104 \\
 B(a_9) &= 0 \\
 GT(a_9, RE_{a_9}) &= 0 \\
 RE_{a_9} &= c_{a_9} + DelayA(a_9, RE_{a_9}) + DelayP(a_9, RE_{a_9}) + B(a_9) + GT(a_9, RE_{a_9}) = 10 + \\
 &14 + 104 = 128 \leq L_{a_2} = 259
 \end{aligned}$$

25

Since the response time of every A-h-k-a process is less than or equal to its deadline, it is thus guaranteed that they are all schedulable.

Below, the second method which uses simulation will be described to determine the worst-case response time of an A-h-k-a process in a feasible pre-run-time



schedule of guaranteed periodic processes, which consists of an initial part of the pre-run-time schedule  $S_0(t_0)$ , which is a subschedule in the interval  $(0, t_0)$ ; and a repeating part of the pre-run-time schedule

5  $S_{LCM}(\text{prd}_{LCM})$ , which is a subschedule in the interval  $(t_0, t_0 + \text{prd}_{LCM})$ .

This method uses the functions of the A-h-k-a Scheduler and the Main Run-Time Scheduler, which are described in below in the section related to the run-

10 time phase.

A method for computing the worst case response time of an A-h-k-a process  $a_i$  in a feasible pre-run-time schedule of guaranteed periodic processes consisting of an initial part of the pre-run-time schedule  $S_0(t_0)$ , in

15 the interval  $(0, t_0)$ ; and a repeating part of the pre-run-time schedule  $S_{LCM}\text{prd}_{LCM}$ , in the interval  $(t_0, t_0 + \text{prd}_{LCM})$  follows:

fail:= false;

20 for  $t_s := 0$  to  $t_0 + \text{prd}_{LCM} - 1$  do  
begin

For each A-h-k-a process  $a_b$ , such that  $a_b \in \text{A-h-k-a} \wedge L_{ab} > L_{ai}$

$\wedge \exists x_k, x_k \in \text{A-h-k-a} \wedge x_k \in P\text{-g}: a_b \text{ excludes } x_k \wedge L_{xk} \leq L_{ai}$ , do the following:

25

let  $a_b$  arrive at time  $t_s - 1$ , and use the A-h-k-a Scheduler and Main Run-Time Scheduler to schedule  $a_b$  and  $a_i$  to obtain a response time of  $a_i$ , called  $RE_1(a_i, t_s, a_b)$ , corresponding to each  $a_b$  according to the

30 assumptions (1)-(6) below, with the additional assumption that  $a_b$  can always start its execution at



end of its time slot in the pre-run-time schedule that was computed using adjusted computation times.

- (1)  $a_i$  arrives in the system at time  $t_s$ .
- 5 (2) Unless  $a_i$  is delayed because of the conditions in Case 1 or Case 5 of the A-h-k-a Scheduler, (see (3) below), let every other A-h-k-a process  $a_j$ , such that  $L_{a_j} \leq L_{a_i}$ , arrive at the following instants in time:  
 $R_{a_j}(k) = t_s + k \cdot \min_{a_j}$ ,  $k = 0, 1, 2, \dots$ , and
- 10 be scheduled before  $a_i$  whenever  $a_i$  and  $a_j$  have both arrived and  $a_i$  has not yet started. If  $a_j$  is delayed by any of the conditions in Cases 1 - 7 of the A-h-k-a Scheduler, then let  $a_i$  be delayed by the same amount of time.
- 15 (All other A-h-k-a processes whose deadlines are shorter or equal to  $a_i$ 's deadline arrive at the same time as  $a_i$  at time  $t_s$ , and are put into execution before  $a_i$ .)
- (3) Whenever the conditions in Case 1 or Case 5 of the
- 20 A-h-k-a Scheduler become true for  $a_i$  and some P-h-k process  $p$  at some time  $t$ , i.e.: if  $\exists p, p \in P-g$ :

P-g:

$$\begin{aligned} & s'(p) \geq t \wedge (a_i \text{ cannot be preempted by } p) \wedge L_{a_i} \geq L_p \\ & \wedge (\nexists p_1, p_1 \in P-g: s(p_1) < s(p) < e(p) < e(p_1) \wedge L_{p_1} \leq L_{a_i}) \\ & \wedge (\nexists p_1, p_1 \in P-g: s(p) < s(p_1) < e(p_1) < e(p) \wedge (a_i \text{ cannot be preempted by } p_1)) \\ & \wedge (\nexists p_m, p_m \in P-g: t < s'(p_m) \wedge e(p_m) \leq s(p) \wedge L_{a_i} < L_{p_m}) \\ & \wedge (e(p) - t) < c_{a_i} + \sum_{p_l \in P-g \wedge t \leq s(p_l) < e(p) \wedge \neg(e'(p_l) \leq t) \wedge L_{p_l} \leq L_{a_i}} c_{p_l} + \\ & \sum_{a_k \in A-h-k-a \wedge d_{a_k} < d_p \wedge (\neg(0 \leq R'(a_k)) \vee (R'(a_k) \leq t \wedge \neg(e'(a_k) \leq t)) \vee (R'(a_k) + \min_{a_k} < e(p)))} \\ & \left\lceil \frac{e(p) - t}{\min_{a_k}} \right\rceil \cdot c_{a_k} \end{aligned}$$

OR

if  $\exists p, p_1, p, p_1 \in P-g$ :

$$s'(p) \geq t \wedge s(p) < s(p_1) < e(p_1) < e(p) \wedge (a_i \text{ cannot be preempted by } p_1) \wedge L_{a_i} \geq L_p$$

$$\begin{aligned} & \wedge (\exists p_m, p_m \in P-g: t < s'(p_m) \wedge e(p_m) \leq s(p) \wedge L_{a_i} < L_{p_m}) \\ & \wedge (s(p) - t) < c_{a_i} + \sum_{p_l \in P-g} \neg(e'(p_l) \leq t) \wedge t \leq s(p_l) < e(p_l) \leq s(p) c_{p_l} \\ & + \sum_{a_k \in A-h-k-a} \wedge L_{a_k} < L_p \wedge (\neg(0 \leq R'(a_k)) \vee (R'(a_k) \leq t \wedge \neg(e'(a_k) \leq t)) \vee (R'(a_k) + \min_{a_k} < s(p))) \\ & \left\lceil \frac{s(p)-t}{\min_{a_k}} \right\rceil \cdot c_{a_k} \end{aligned}$$

Let  $t$  also be the earliest time that the conditions in Case 1 or Case 5 become true for that instance of  $p$  and  $a_i$ , then for every other  $A-h-k-a$  process  $a_j$  that belongs to the following set:

$$\begin{aligned} & \{a_j | a_j \in A-h-k-a \wedge j \neq i \wedge L_{a_j} \leq L_{a_i} \wedge \\ & ((\neg(0 \leq R'(a_j)) \vee (R'(a_j) \leq t \wedge \neg(e'(a_j) \leq t)) \vee (R'(a_j) + \min_{a_j} < e(p))) \\ & \vee (s'(a_b) \leq t_s \wedge t-1 \leq e'(a_b) \wedge s'(a_j) < t))\} \end{aligned}$$

( $a_j$  has a deadline that is shorter than or equal to  $a_i$ 's deadline and, either  $a_j$  has never arrived; or  $a_j$  has arrived but has not completed its computation; or  $a_j$  may arrive again before time  $s(p)$ ; or at time  $t-1$   $a_i$  was blocked by  $a_b$  and  $a_j$  started its execution before  $t$ ), "delay"  $a_j$ 's arrival time to occur at time  $s(p)$ ; if at time  $s(p)$  the conditions in case 1 or Case 5 of the  $A-h-k-a$  Scheduler continue to be true for some other periodic process  $p'$ , then  $a_i$  should be delayed again, and the  $A-h-k$  processes  $a_j$  that satisfy the conditions of the formula for  $p'$  should be delayed again until  $s(p')$ ; otherwise  $a_j$  arrives at  $s(p)$ , and is scheduled before  $a_i$ ; and assume that  $p$  starts executing at time  $s(p) + \sum_{a_k \in A-h-k-a} \wedge L_{a_k} < L_p \} c_{a_k}$ .

For each such  $a_j$ , let only a single instance of  $a_j$  arrive at time  $s(p)$ , even if originally there could be several instances of a same process  $a_j$  that satisfy the conditions above.

- 5           For each such  $a_j$ 's subsequent arrival times after  $s(p)$ , use the same arrival times that are determined in (2), i.e.,  $a_j$ 's subsequent arrival times after  $s(p)$  will be:

$$R_{a_j}(k) = t_s + k \cdot \min_{a_j} \text{ such that } R_{a_j}(k) > s(p).$$

- 10           (If at time  $t$  there exists more than one process  $p$  for which the conditions of either Case 1 or Case 5 are true for  $p$  and  $a_i$ , then let the above apply to the process  $p$  among them that has the latest  $s(p)$  time in the pre-run-time schedule.)

- 15           (If  $a_i$  is delayed due to the conditions in Case 1 or Case 5, then an A-h-k-a process  $a_j$  could delay  $a_i$  by a maximum amount by arriving at  $s(p)$ .)

- (4) If the end of the current instance of the repeating part of the pre-run-time schedule is reached,  
20           continue at the beginning of the next instance of the repeating part of the pre-run-time schedule.

            (5) If  $a_i$ 's deadline  $d_{a_i}$  is reached but  $a_i$  has not yet completed its computation, then set  $fail := true$  and exit from the procedure.

- 25           (6) If  $a_i$ 's computation is completed before its deadline  $d_{a_i}$ , then record the completion time of  $a_i$  as the response time of  $a_i$  (either  $RE_1(a_i, t_s, a_b)$  for the current  $a_b$ , or  $RE_2(a_i, t_s)$  when no such  $a_b$  is assumed to have arrived at time  $t_s - 1$ ).

- 30           After the response time  $RE_1(a_i, t_s, a_b)$  corresponding to every  $a_b$  that may block  $a_i$  has been

computed, and the response time  $RE_2(a_i, t_s)$  that assumed that no such  $a_b$  had arrived and blocked  $a_i$  has also been computed, set the response time of  $a_i$  with arrival time  $t_s$ ,  $RE(a_i, t_s)$  to be equal to the greatest among them, 5 i.e.,  $RE(a_i, t_s) = \max\{RE_1(a_i, t_s, a_b), RE_2(a_i, t_s)\}$ , and exit from the current iteration of the loop for this value of  $t_s$ , and start the next iteration for  $t_s := t_s + 1$ .

10 end;

if not fail then

$RE_{ai} := \max\{RE(a_i, t_s) \mid t_s = 0, 1, \dots, LCM - 1\}$

A description will be found below of scheduling 15 of the A-h-k-a processes, of the A-h-k-a Scheduler and of definitions of the terms used above.

It is assumed in a second assumption that for each A-h-k-a process  $a_i$ , for each  $RE_{ai}$  determined in the above procedure, if  $RE_{ai} \Rightarrow L_{ai}$ ,  $a_i$  will always meet its 20 deadline at run-time.

### Example 7

If the second method is used, that is, the 25 simulation procedure above, to determine the response times of the A-h-k-a processes in Examples 1-6 above, the procedure will find the following.

$RE_{a0}$ :

The maximum value of  $RE(a_0, t_s)$  will occur when 30  $a_0$  arrives at time  $t_s = 0$ . Since no process excludes  $a_0$ , and  $a_0$  has the minimum latitude among all processes,  $a_0$

will always be put into execution immediately after it arrives, thus  $a_0$ 's response time  $RE_{a0} = RE(a_0, 0) = \max\{RE(a_0, t_s)\} = c_{a0} = 2 \Rightarrow L_{a0} = 2$ .

5  $RE_{a1}$ :

The maximum value of  $RE(a_1, t_s)$  will occur when  $a_1$  arrives at time  $t_s = 0$ . Since no process excludes  $a_1$ , and only one process  $a_0$  has less latitude compared with  $a_1$ 's latitude, when  $a_1$  arrives at time  $t_s = 0$ , assuming  
 10 that  $a_0$  will also arrive at time  $t_s = 0$ ,  $a_1$  will only be delayed by  $a_0$ 's execution time, thus  $a_1$ 's response time  $RE_{a1} = \max\{RE(a_1, t_s)\} = RE(a_1, 0) = c_{a0} + c_{a1} = 2 + 2 = 4 \Rightarrow L_{a1} = 7$ .

15  $RE_{a2}$ :

The maximum value of  $RE(a_2, t_s)$  will occur when  $a_2$  arrives at time  $t_s = 82$  but  $a_9$  arrived one time unit earlier at  $t_s - 1 = 81$ , so  $a_9$  blocks  $a_2$  at time 82. According to rule (2) in the simulation procedure, it is  
 20 first assumed that  $a_0$  and  $a_1$  will also arrive at time  $t_s = 82$ , and will be put into execution from time 82 to 86, preempting  $a_9$ . After  $a_0$  and  $a_1$ 's completion,  $a_9$  will resume at time 86 and complete its computation at time 95. At time 95,  $a_2$  will be delayed by the conditions of  
 25 Case 1 of the A-h-k-a Scheduler, because  $e(newp_3) - t = 114 - 95 = 19 < c_{a2} + c_{newp_3} = 10 + 10 = 20$ . According to rule (3) in the simulation procedure, since at time 95  $a_0$  and  $a_1$  belong to the set:

$$\{ a_j \mid a_j \in A-h-k-a \wedge j \neq i \wedge L_{aj} \leq L_{ai} \wedge ((\bigwedge (0 \leq R'_{aj})) \vee (R'_{aj} \leq t \wedge \bigwedge (e'_{aj} \leq t)) \vee (R'_{aj} + \min_{aj} < e(p)))$$

30

$$\wedge (s'(a_b) \leq t_s \wedge t - 1 \leq e'(a_b) \wedge s'_{a_j} < t))\}$$

because  $a_0 \in A\text{-h-k-a} \wedge a_0 \neq a_2 \wedge d_{a_0} \leq d_{a_2} \wedge (s'(a_9) \leq t_s$   
 $= 82 \wedge t - 1 = 95 - 1 = 94 \leq e'_{a_9} = 95 \wedge s'(a_0) = 82 <$   
 $t = 95.$

5 (At time  $t - 1 = 95 - 1 = 94$   $a_2$  was blocked by  $a_9$   
 ("a<sub>b</sub>"), and  $a_0$  ("a<sub>j</sub>") started its execution before  $t =$   
 95).  $a_1$  also meets the above conditions similar to  $a_0$ .  
 According to rule (3) in the simulation procedure,  $a_0$   
 and  $a_1$ 's arrival times are "delayed" to occur at time  
 10  $s(\text{newp}_3) = 100$  and are scheduled before  $a_2$  to execute  
 from time 100 to 104;  $\text{newp}_3$  starts executing at time  
 $s(\text{newp}_3) + c_{a_0} + c_{a_1} = 100 + 2 + 2 = 104$ , and completes  
 its execution at time 114.

As  $L_{p_6} = 200 < L_{a_2} = 239$  and  $L_{p_7} = 200 < L_{a_2} =$   
 15 239,  $p_6$  will execute from 114 to 140;  $p_7$  will execute  
 from 140 to 156;  $a_2$  will execute from 156 to 166. Thus  
 $a_2$ 's response time  $RE_{a_2} = \max\{RE(a_2, t_s)\} = RE(a_2, 82) =$   
 $e'(a_2) - R_{a_2} = 166 - 82 = 84 < L_{a_2} = 239.$

In effect, the simulation procedure computes a  
 20 response time of  $a_2$  that is equal to  $a_2$ 's response time  
 in the case that is illustrated in Fig. 2, where  $a_9$   
 arrives at time 81,  $a_2$  arrives at time 82 and is blocked  
 by  $a_9$  until time 91; at time 91  $a_2$  is delayed by the  
 conditions of Case 1 of the A-h-k-a Scheduler, because  
 25  $e(\text{newp}_3) - t = 114 - 91 = 23 < c_{a_2} + c_{\text{newp}_3} + c_{a_0} + c_{a_1} =$   
 $10 + 10 + 2 + 2 = 24.$

$a_0$  and  $a_1$  arrive at time  $s(\text{newp}_3) = 100.$

$RE_{a_9}:$

30 The maximum value of  $RE(a_9, t_s)$  will occur when  
 $a_9$  arrives at time  $t_s = 100$ ;  $a_0$  and  $a_1$  also arrive at





asynchronous, by setting the threshold values in the procedure for converting a set of asynchronous processes into periodic processes given earlier.

Assume that the value chosen for the threshold  
 5 for  $a_A$  was large and asynchronous process  $a_A$  was not converted into a new periodic process. The procedure for scheduling a set of periodic processes with offsets will construct the pre-run-time schedule for the processes  $p_B$ ,  $p_C$ ,  $p_D$  illustrated in Figure 19. Assuming  
 10  $a_A$  is to be scheduled by the A-h-k-a Scheduler using the information in the pre-run-time schedule, the simulation procedure will determine that the worst case response time of  $a_A$  will happen when process  $a_A$  makes a request at time 6. At time 6, process  $a_A$  will be delayed by the  
 15 conditions in Case 1 of the A-h-k-a scheduling method.  $a_A$  will be executed from time 11 to time 13 after  $p_C$  has completed. The response time of process  $a_A$  is  $RE(a_A, t_s) = RE(a_A, 6) = e'(a_A) - s'(a_A) = 13 - 6 = 7 \leq d_{aA} = 7$ . In this case process  $a_A$  is also guaranteed to always meet  
 20 its deadline.

The simulation procedure above is more accurate than the method described earlier for determining the worst case response times of A-h-k-a processes, because the simulation procedure takes into account knowledge  
 25 about the positions of the periodic processes in the pre-run-time schedule. In contrast, the earlier formula does not take into account such knowledge, and assumes that in the worst case, all periodic processes may arrive at the same time. Note that currently, none of  
 30 the algorithms and protocols in the prior art that perform all scheduling activities at run-time, can

avoid making the latter overly pessimistic assumption in their schedulability tests.

If it is determined that the deadline of some hard deadline process cannot be met, that is, if a  
5 feasible pre-run-time schedule does not exist for the given set of P-h-k processes, or if the response time of some A-h-k-a process exceeds its deadline, then one may have to modify the characteristics of or remove one or more P-h-k or A-h-k-a processes.

10 In the first case, the pre-run-time scheduling method will identify the critical set, that is, the subset of P-h-k processes for which either modifying the characteristics of one or more processes in that set, or removing one or more processes from that set is  
15 necessary to meet the deadlines of all the P-h-k processes.

In the second case, the formula or simulation procedure for calculating the response time of each A-h-k-a process can be used to identify which processes  
20 should be modified or removed, in order to meet the deadlines of all the A-h-k-a processes.

After the user has modified the characteristics of, or removed one or more P-h-k or A-h-k-a processes, the method will be applied again. The initial  
25 determination and any subsequent modification of process characteristics by the user depends on the specific user application requirements and is outside the scope of this specification. This process must be repeated until there exists a feasible schedule for all  
30 the hard-deadline processes.

Step 4: A feasible pre-run-time schedule for the P-s-k and P-h-k processes is constructed.

After guaranteeing the schedulability of all the processes

5 with hard deadlines, the set of periodic processes with soft deadlines and known characteristics (P-s-k processes) is scheduled together with the set of all periodic processes with hard deadlines and known characteristics (P-h-k processes), a feasible pre-run-  
10 time schedule for these processes is constructed. Each P-s-k process is assigned an integer, called its "criticality level." Each P-s-k process is also assigned, in addition to its normal deadline, a "deadline upper-limit."

15 The computation times of the soft-deadline periodic P-s-k processes are modified in the same way as for the hard-deadline periodic P-h-k processes.

If it is determined that a feasible schedule does not exist, then the method will identify the soft  
20 critical set, that is, the subset of soft-deadline processes for which either modifying the characteristics of one or more processes in that set, or removing one or more processes from that set, is necessary to meet the deadlines of all hard-deadline processes. The method  
25 will select one process that has the lowest criticality level among the processes in the soft-critical set and increase the deadline of that process by an amount that does not exceed the deadline upper-limit of that process and attempt to find a feasible schedule again. The  
30 deadline of the process with the lowest criticality level will be incremented until one of the following

happens: either a) a feasible schedule is found; or b) the previously selected process does not belong to the newly computed critical set; or c) the revised deadline of that process cannot be further increased without  
5 exceeding the deadline upper-limit.

In the latter two cases, the method will select another process that has the lowest criticality level among all processes in the soft critical set and for which the deadline has not yet been revised, increment  
10 its deadline, and attempt to find a feasible schedule again.

If it is determined that a feasible schedule still does not exist after the deadlines of all the processes in the soft critical set have been revised and  
15 their deadline upper-limits have been reached, the method will provide the user with the list of soft-deadline processes in the soft critical set for which modifying the characteristics of one or more processes in that set or removing one or more processes in that  
20 set is necessary to meet the deadlines of all hard-deadline processes.

After the user has modified the characteristics of one or more processes in the soft critical set, or removed one or more processes in that set, the method  
25 will be applied again. The procedure will be repeated until there exists a feasible schedule for all the P-h-k and P-s-k processes. Again, the initial determination and any subsequent modification of process characteristics by the user depends on the specific user  
30 application requirements and is outside the scope of this specification.

After a feasible schedule has been found for all the P-h-k and P-s-k processes, the worst-case response times for all the A-h-k-a processes are recomputed with the set of P-s-k processes included in the set of  
5 guaranteed periodic processes P-g. If the worst-case response time of any A-h-k-a process  $a_i$  exceeds its deadline, the method will select one process that has the lowest criticality level among all P-s-k processes that contribute to  $a_i$ 's worst-case response time and  
10 increase the deadline of that P-s-k process by an amount that does not exceed its deadline upper-limit and recompute the worst-case response time. The deadline of the process with the lowest criticality level will be incremented until one of the following happens: either  
15 a) the worst-case response time of every A-h-k-a process is less than or equal to its deadline; or b) the previously selected process does not belong to the newly computed set of P-s-k processes that contribute to  $a_i$ 's worst-case response time; or c) the revised deadline of  
20 that process cannot be further increased without exceeding the deadline upper-limit. In the latter two cases, the method will select another process that has the lowest criticality level among all P-s-k processes that contribute to  $a_i$ 's worst-case response time and for  
25 which the deadline has not yet been revised, increment its deadline, and compute the worst-case response time again for each A-h-k-a process.

If it is determined that the worst-case response time of some A-h-k-a process still exceeds its deadline  
30 after the deadlines of all the processes that contribute to  $a_i$ 's worst-case response time have been revised and

their deadline upper-limits have been reached, the method will provide the user with the list of soft-deadline processes in the set that contribute to  $a_i$ 's worst-case response time for which modifying the

5 characteristics of one or more processes in that set or removing one or more processes in that set is necessary to meet the deadlines of all the A-h-k-a processes.

After the user has modified the characteristics of one or more processes that contribute to  $a_i$ 's worst-

10 case response time, or removed one or more processes in that set, the method will be applied again. The procedure should be repeated until the worst-case response time of every A-h-k-a process is less than or equal to its deadline.

15 At the end of Step 4 the method will generate a feasible pre-run-time schedule for all the P-h-k, and P-s-k processes, while guaranteeing that the response times of all the A-h-k-a processes will be less than or equal to their deadlines.

20 The set of P-h-k and P-s-k processes will be referred to as the set of guaranteed periodic processes. A simplified procedure for implementing Step 4 will be described later.

## 25 Example 8

Assume that in addition to the hard deadline processes described in Examples 1-7 above, the following periodic process has a soft deadline and known

30 characteristics (P-s-k process):

$p_8: R_{p_8} = 20, c_{p_8} = 16, d_{p_8} = 55, prd_{p_8} = 200; .$

Assume further that  $p_8$ 's criticality is 3, its deadline upperlimit is 100.

$p_8$ 's adjusted computation time is:

$$5 \quad c_{p8}' = c_{p8} + ca_0 + ca_1 = 16 + 2 + 2 = 20.$$

Firstly it will be determined that no feasible schedule exists for the given set of process parameters. The optimal schedule for the given set of parameters is the same as the schedule shown in Figure 3, except that  
 10  $d_{p8} = 55$  and  $p_8$  is late by 5 time units. The soft critical set contains one process  $p_8$ . The simplified procedure for Step 4 referred to above will increase the deadline of  $p_8$  until  $d_{p8}' = 60$ , at which point the feasible schedule of the P-h-k and P-s-k processes is  
 15 obtained, as shown in Figure 3.

If the formula in the first method is used to calculate the response times of the A-h-k-a processes the following is obtained:

$RE_{a_0}, RE_{a_1}$  remain the same as in Step 3, and are less than  $L_{a_0}$  and  $L_{a_1}$  respectively.

$$\text{DelayA}(a_2, RE_{a_2}) = \lceil RE_{a_2}/\min_{a_0} \rceil c_{a_0} + \lceil RE_{a_2}/\min_{a_1} \rceil c_{a_1} = 2 + 2 = 4$$

$$\begin{aligned} \text{DelayP}(a_2, RE_{a_2}) &= \lceil RE_{a_2}/prd_{newp_3} \rceil c_{newp_3} + \lceil RE_{a_2}/prd_{p_4} \rceil c_{p_4} + \lceil RE_{a_2}/prd_{p_5} \rceil c_{p_5} \\ &+ \lceil RE_{a_2}/prd_{p_6} \rceil c_{p_6} + \lceil RE_{a_2}/prd_{p_7} \rceil c_{p_7} + \lceil RE_{a_2}/prd_{p_8} \rceil c_{p_8} \\ &= 20 + 26 + 16 + 26 + 16 + 16 = 120 \end{aligned}$$

$$B(a_2) = c_{a_2} = 10;$$

$$GT(a_2, RE_{a_2}) = \lceil RE_{a_2}/prd_{newp_3} \rceil c_{a_2} = 20$$

$$RE_{a_2} = c_{a_2} + \text{DelayA}(a_2, RE_{a_2}) + \text{DelayP}(a_2, RE_{a_2}) + B(a_2) + GT(a_2, RE_{a_2}) = 10 + 4 + 120 + 10 + 20 = 164 \leq L_{a_2} = 239$$

$$\begin{aligned} \text{DelayA}(a_9, RE_{a_9}) &= \lceil RE_{a_9}/\min_{a_0} \rceil c_{a_0} + \lceil RE_{a_9}/\min_{a_1} \rceil c_{a_1} + \lceil RE_{a_9}/\min_{a_2} \rceil c_{a_2} = 2 + 2 \\ &+ 10 = 14 \end{aligned}$$



$$\begin{aligned} \text{DelayP}(a_9, RE_{a_9}) &= \lceil RE_{a_9}/prd_{newp_3} \rceil c_{newp_3} + \lceil RE_{a_9}/prd_{p_4} \rceil c_{p_4} + \lceil RE_{a_9}/prd_{p_5} \rceil c_{p_5} \\ &+ \lceil RE_{a_9}/prd_{p_6} \rceil c_{p_6} + \lceil RE_{a_9}/prd_{p_7} \rceil c_{p_7} + \lceil RE_{a_9}/prd_{p_8} \rceil c_{p_8} \\ &= 20 + 26 + 16 + 26 + 16 + 16 = 120 \end{aligned}$$

$$B(a_9) = 0;$$

$$GT(a_9, RE_{a_9}) = 0;$$

$$RE_{a_9} = c_{a_9} + \text{DelayA}(a_9, RE_{a_9}) + \text{DelayP}(a_9, RE_{a_9}) + B(a_9) + GT(a_9, RE_{a_9}) = 10 + 14 + 120 = 144 \leq L_{a_9} = 239$$

If the second method is used, that is, the simulation procedure described above, to calculate the response times of the A-h-k-a processes in the examples above, the procedure will find the following:

- 10  $a_0$ 's response time  $RE_{a_0}$  and  $a_1$ 's response time  $RE_{a_1}$  will remain the same as in Example 6, that is:

$$RE_{a_0} = RE(a_0, 0) = \max\{RE(a_0, t_s)\} = c_{a_0} = 2 \leq d_{a_0} = 2.$$

$$\begin{aligned} RE_{a_1} &= \max\{RE_{a_1}, t_s\} = RE(a_1, 0) = c_{a_0} + c_{a_1} = 2 + 2 = 4 \\ &\leq d_{a_1} = 7. \end{aligned}$$

- 15  $RE_{a_2}$ :

The maximum value of  $RE(a_2, t_s)$  will happen when  $a_2$  arrives at time  $t_s = 182$  but  $a_9$  arrived one time unit earlier at  $t_s - 1 = 181$ , so  $a_9$  blocks  $a_2$  at time 182.

- 20 According to rule (2) in the simulation procedure, it is first assumed that  $a_0$  and  $a_1$  will also arrive at time  $t_s = 182$ , and will be put into execution from time 182 to 186, preempting  $a_9$ . After  $a_0$  and  $a_1$ 's completion,  $a_9$  will resume at time 186 and complete its computation at time 195. At time 195,  $a_2$  will be delayed
- 25 by the conditions of Case 1 of the A-h-k-a Scheduler, because  $e(newp_3 - t) = 214 - 195 = 19 < c_{a_2} + c_{newp_3} = 10 + 10 = 20$ .

- 30 According to rule (3) in the simulation procedure, since at time 195  $a_0$  and  $a_1$  belong to the set:

$\{a_j | a_j \in A\text{-h-k-a} \wedge j \neq i \wedge L_{a_j} \leq L_{a_i} \wedge$   
 $((\neg(0 \leq R'(a_j)) \vee (R'(a_j) \leq t \wedge \neg(e'(a_j) \leq t))) \vee (R'(a_j) + \min_{a_j} < e(p)))$   
 $\vee (s'(a_b) \leq t_s \wedge t-1 \leq e'(a_b) \wedge s'(a_j) < t))\}$   
 because  $a_0 \in A\text{-h-k-a} \wedge a_0 \neq a_2 \wedge L_{a_0} \leq L_{a_2} \wedge (s'(a_0) \leq t_s = 182 \wedge t-1 = 195-1 =$   
 $194 \leq e'(a_0) = 195 \wedge s'(a_0) = 182 < t = 195.$

(at time  $t - 1 = 195 - 1 = 194$   $a_2$  was blocked by  
 $a_9$  (" $a_b$ ") and  $a_0$  (" $a_j$ ") started its execution before  $t =$   
 195).  $a_1$  also meets the above conditions similar to  $a_0$ .

10 According to rule (3) in the simulation  
 procedure,  $a_0$  and  $a_1$ 's arrival times are "delayed" to  
 occur at time  $s(\text{newp}_3) = 200$  and are scheduled before  $a_2$   
 to execute from time 200 to 204;  $\text{newp}_3$  starts executing  
 at time  $s(\text{newp}_3) + c_{a_0} + c_{a_1} = 200 + 2 + 2 = 204$ , and

15 completes its execution at time 214;  
 $a_2$  starts at time 214 and executes until time 220.  
 As  $L_{p_8} = d_{p_8}' - r_{p_8} = 260 - 220 = 40 < L_{a_2} = d_{a_2} = 239$ ,  $L_{p_5}$   
 $= d_{p_5} - r_{p_5} = 250 - 230 = 20 < L_{a_2} = d_{a_2} = 239$ ,  
 and  $L_{p_4} = d_{p_4} - r_{p_4} = 200 - 0 = 200 < L_{a_2} = d_{a_2} = 239$ ;

20  $p_8$  will preempt  $a_2$  at time 220;  
 $p_8, p_5, p_4$  will execute from time 220 to 278;  
 $a_2$  will resume execution from time 278 to 282;  
 Thus  $a_2$ 's response time  $RE_{a_2} = \max\{RE(a_2, t_s)\} = RE(a_2,$   
 $182) = e'(a_2) - R_{a_2} = 282 - 182 = 100 < L_{a_2} = 239.$

25 Figure 4 illustrates a possible run-time  
 executionn of the A-h-k-a processes  $a_0, a_1, a_2, a_9$ ,  
 together with the P-h-k and P-s-k processes of Figure 3.  
 The worst case response time of the A-h-k-a process  $a_2$   
 is  $RE_{a_2} = \max\{RE(a_2, t_s)\} = RE(a_2, 182) = e'(a_2) - R_{a_2} = 282$   
 30  $- 182 = 100$ , as computed in step 4.

In effect, the simulation procedure computes a worst-case response time of  $a_2$  that is equal to the response time of  $a_2$  in the case that is illustrated in Figure 4, where  $a_9$  arrives at time 181,  $a_2$  arrives at time 182 and is blocked by  $a_9$  until time 191; at time 191  $a_2$  is delayed by the conditions of Case 1 of the A-h-k-a Scheduler, because  $e(\text{newp}_3) - t = 214 - 191 = 23 < C_{a2} + C_{\text{newp}_3} + C_{a0} + C_{a1} = 10 + 10 + 2 + 2 = 24$ .  $a_0$  and  $a_1$  arrive at time  $s(\text{newp}_3) = 200$ .

10  $RE_{a9}$ :

The maximum value of  $RE(a_9, t_s)$  will happen when  $a_9$  arrives at time  $t_s = 200$ ;  $a_0$  and  $a_1$  also arrive at time 200 and execute from time 200 to 204;  $\text{newp}_3$  executes from time 204 to 214;  $a_2$  starts at time 214 and executes until time 220.

As  $L_{p8} < L_{a2}$ ,  $L_{p5} < L_{a2}$ , and  $L_{p4} < L_{a2}$ ;  $p_8$  will preempt  $a_2$  at time 220;  $p_8, p_5, p_4$  will execute from time 220 to 278;  $a_2$  will resume execution from time 278 to 282;

Finally  $a_9$  will execute from time 282 to 292.

20 Thus  $a_9$ 's response time  $RE_{a9} = \max\{RE(a_9, t_s)\} = RE(a_9, 100) = e'(a_9) - Ra_9 = 292 - 200 = 92 < L_{a9} = 259$ .

Since the response time of every A-h-k-a process is less than or equal to its deadline, it thus can be guaranteed that they are all schedulable. Note again that the response time of  $a_2$  determined by the simulation procedure is 100 and is tighter than the response time of 164 determined by the formula in the first method.

The response time of  $a_9$  determined by the simulation procedure is 92 and also is tighter than the

response time of 144 determined by the formula in the first method.

Fig. 3 illustrates the feasible pre-run-time schedule in which each guaranteed periodic process reserves a time frame that includes reserved processor capacity for any A-h-k-a process that has a shorter deadline than that guaranteed periodic process's deadline.

10 Step 5: Determine the response times of the A-s-k processes

A-s-k processes are scheduled at run-time by the Main Run-Time Scheduler, but their worst-case response times are determined before run-time in this step. A-s-k processes are scheduled at a lower priority than the A-h-k-a, P-h-k, P-s-k processes. That is, A-s-k processes are executed only when there does not exist any A-h-k-a, P-h-k, or P-s-k process that is ready for execution. If more than one A-s-k process are competing for execution, the process with the shortest deadline will be chosen for execution.

An A-s-k process is not allowed to have any precedence relation with any other process. An A-s-k process  $a_i$  is also not allowed to have any exclusion relation of the form  $a_i$  excludes  $x$  where  $x$  is an A-h-k-a, P-h-k, or P-s-k process. These restrictions are imposed in order to prevent A-s-k processes from causing any delay to A-h-k-a, P-h-k, or P-s-k processes, so that one can guarantee that all the hard deadlines will be satisfied, and also provide firm response time

guarantees for all the processes with known characteristics.

Two different methods of determining the worst-case

5 response time of an A-s-k process will now be described.

The first method uses a mathematical formula to calculate the response time, and is very similar to the formula described in Step 3 for determining the worst-case response time of an A-h-k-a process, except that:

10 (a) all A-h-k-a, P-h-k and P-s-k processes have higher priority than any A-s-k process  $a_i$ , so their execution times are included together with the execution times of A-s-k processes that have shorter deadlines than  $a_i$  in the terms  $\text{Delay}(a_i, RE_{ai})$  and  $\text{Delay}(a_i, RE_{ai})$ ;

15 (b) because an A-s-k process cannot exclude a P-h-k, P-s-k, or

A-h-k-a process, the term  $GT(a_i, RE_{ai})$  is not required in the formula; and (c) the term  $B(a_i)$  only needs to

20 consider the blocking time due to A-s-k processes that have greater deadlines than  $a_i$ . Because the rest of the formula in Step 3 is basically the same as the formula used here, to avoid repetition, the complete definition of the formula used here is provided in Appendix 3.

25

#### Example 9

Assume the following asynchronous process with a soft deadline and known characteristics (A-s-k process):

30

$a_{10}: C_{a_{10}} = 10, d_{a_{10}} = 300, \min_{a_{10}} = 300.$

The procedure to be described below for  
determining the response times of the A-s-k processes is  
used to determine the response time of the A-s-k process  
5  $a_{10}$ , the following will be processed:

$$\begin{aligned} \text{DelayA}(a_{10}, RE_{a_{10}}) &= \lceil RE_{a_{10}} / \min_{a_0} \rceil c_{a_0} + \lceil RE_{a_{10}} / \min_{a_1} \rceil c_{a_1} + \lceil RE_{a_{10}} / \min_{a_2} \rceil c_{a_2} \\ &+ \lceil RE_{a_{10}} / \min_{a_9} \rceil c_{a_9} = 2 + 2 + 10 + 10 = 24 \\ \text{DelayP}(a_{10}, RE_{a_{10}}) &= \lceil RE_{a_{10}} / \text{prd}_{\text{newp}_3} \rceil c_{\text{newp}_3} + \lceil RE_{a_{10}} / \text{prd}_{p_4} \rceil c_{p_4} + \lceil RE_{a_{10}} / \text{prd}_{p_5} \rceil c_{p_5} \\ &+ \lceil RE_{a_{10}} / \text{prd}_{p_6} \rceil c_{p_6} + \lceil RE_{a_{10}} / \text{prd}_{p_7} \rceil c_{p_7} + \lceil RE_{a_{10}} / \text{prd}_{p_8} \rceil c_{p_8} = 20 + 26 + 16 + 26 + 16 \\ &+ 16 = 120 \\ B(a_{10}) &= 0 \\ RE_{a_{10}} &= c_{a_{10}} + \text{DelayA}(a_{10}, RE_{a_{10}}) + \text{DelayP}(a_{10}, RE_{a_{10}}) = 10 + 24 + 120 = 154 \\ &\leq L_{a_{10}} = 300 \end{aligned}$$

The second method uses a simulation procedure to  
15 determine the worst-case response time of each A-s-k  
process. The simulation procedure used here is also  
very similar to the simulation procedure described in  
Step 3 for determining the worst-case response time of  
an A-h-k-a process, except that,  
20 (a) because all A-h-k-a processes have higher priority  
than any A-s-k process  $a_i$ , they are all assumed to  
arrive at the same time as  $a_i$ , together with all A-s-k  
processes that have shorter deadlines compared with  $a_i$ ;  
(b) because an A-s-k process cannot exclude a P-h-k, P-  
25 s-k or A-h-k-a process, there is no need for an A-s-k  
process to avoid blocking a P-h-k, P-s-k or A-h-k-a  
process such as in Case 1 and Case 5 of the A-h-k-a  
Scheduler; consequently there is no need for a rule in  
the simulation procedure for A-s-k processes that is  
30 similar to the rule (5) in the simulation procedure in  
Step 3.

Because the rest of the simulation procedure in Step 3 is basically the same as the simulation procedure used here, to avoid repetition, the complete description of the simulation procedure used here is given later in this specification.

Similar to the case in Step 3, compared with the formula, the simulation procedure gives tighter response times that can guarantee the schedulability of a larger number of cases, but requires a longer computation time.

The simulation procedure used here also uses the functions of the A-h-k-a Scheduler and the Main Run-Time Scheduler, which are described later.

#### Example 10

This example is a description of the use of this simulation procedure. Assume that the second method, that is, the simulation procedure described later, is used to calculate the worst-case response time of the A-s-k process  $a_{10}$  in Example 9 above, the procedure will find the following:

The maximum value of  $RE(a_{10}, t_s)$  will happen when  $a_{10}$  arrives at time  $t_s = 196$ . Figure 5 is an illustration of this case, a run-time execution of the A-s-k process  $a_{10}$ , together with the A-h-k-a processes  $a_0, a_1, a_2, a_9$ , and the P-h-k and P-s-k processes of Figure 3. The simulation procedure simulates this execution in which the worst case response time of  $a_{10}$ ,  $RE_{a_{10}} = \max\{RE(a_{10}, t_s)\} = RE(a_{10}, 196) = e'(a_{10}) - R_{a_{10}} = 298 - 196 = 102$ , as determined in Example 10, step 5.

At time 196, the simulation procedure assumes that  $a_0, a_1, a_2, a_9$  will arrive at the same time as  $a_{10}$ , so  $a_0$  will execute from time 196 to 198, and  $a_1$  will execute from time 198 to 200.  $newp_3$  will execute from  
5 time 200 to 210.  $a_2$  will execute from time 210 to 220.  $p_8, p_5, p_4$  will execute from time 220 to time 278.  $a_9$  will execute from time 278 to 288.

$a_{10}$  will execute from time 288 to 298. Thus  $a_{10}$ 's worst-case response time will be  $RE_{a_{10}} = \max\{RE(a_{10}, t_s)\}$   
10  $= RE(a_{10}, 196) = e'(a_{10}) - R_{a_{10}} = 298 - 196 = 102 < L_{a_{10}} = 300$ .

#### Run-Time Phase

15 During run-time, the processor capacity that is left unused by guaranteed periodic processes (the set of P-h-k and P-s-k processes P-g) in the pre-run-time schedule generated in Step 4 will be used to schedule the  
20 processes that are asynchronous and for which the characteristics are either known before run-time (A-s-k and A-h-k-a processes) or unknown before run-time but known as soon as the process arrives in the system (A-s-u processes).

25 In the previous step, a time slot in the feasible pre-run-time schedule was reserved for each guaranteed periodic process. However, at run-time each periodic process may not necessarily execute within its time slot in a pre-run-time schedule, because provided  
30 that it can be guaranteed that all the constraints will be satisfied, it is preferred that each process should



be executed as early as possible at run-time, in order to minimize the response times. Nevertheless, the beginning and end times of the time slots are important parameters that will be used by the run-time scheduler to determine, at each point in time, whether each asynchronous or periodic process can be safely put into execution while guaranteeing that all the constraints will be satisfied.

In particular, the run-time scheduler will always guarantee that,

- (1) the actual execution order of each pair of "guaranteed" periodic processes  $p_1$  and  $p_2$  will be the same as the relative ordering of their time slots in the pre-run-time schedule, that is, if  $e(p_1) \leq s(p_2)$ , then  $e'(p_1) \leq s'(p_2)$ ; and
- (2) the actual completion time  $e'(p)$  of each "guaranteed" periodic process  $p$  will never exceed the end of  $p$ 's time slot  $e(p)$  in the pre-run-time schedule.

## 20 Scheduling A-h-k-a Processes

Each time the Run-Time Scheduler is executed, it will first try to schedule A-h-k-a processes according to the procedure below.

25 For any A-h-k-a process  $a_i$  and any P-g process  $p_1$ , the following should hold:

$a_i$  should not be able to be preempted by  $p_1$ , if the following conditions hold:

(  $a_i$  excludes  $p_1 \vee (\exists a_j, a_j \in \text{A-h-k-a}: L_{a_j} < L_{p_1} \wedge a_i$   
 30 excludes  $a_j)$

# Simplified A-h-k-a Scheduler Method

The A-h-k-a Scheduler Method functions as part of the

5 Main-Run-Time Scheduler to be described below.

A-h-k-a-Scheduler:

At any time t:

if some A-h-k-a process  $a_i$  has arrived at time t,  
or if some process  $x_i$  completes its computation at time

10 t

or if t is both the release time and start time in the  
pre-run-time schedule for some P-g process p, i.e.,  $t = R_p = s(p)$

then

15 begin

for each A-h-k-a process  $a_i$  that has already  
arrived and not yet completed,

i.e.,  $R'(a_i) \leq t \wedge \neg(e'(a_i) \leq t)$ , if  $a_i$   
satisfies any of the following conditions,

20 then Delay  $a_i$ :

Case 1:

if  $\exists p, p \in P-g$ :

$s'(p) \geq t \wedge (a_i \text{ cannot be preempted by } p) \wedge L_{a_i} \geq L_p$

$\wedge (\nexists p_l, p_l \in P-g: s(p_l) < s(p) < e(p) < e(p_l) \wedge L_{p_l} \leq L_{a_i})$

$\wedge (\nexists p_l, p_l \in P-g: s(p) < s(p_l) < e(p_l) < e(p) \wedge (a_i \text{ cannot be preempted by } p_l))$

$\wedge (\nexists p_m, p_m \in P-g: t < s'(p_m) \wedge e(p_m) \leq s(p) \wedge L_{a_i} < L_{p_m})$

$\wedge (e(p) - t) < c_{a_i} + \sum_{p_l \in P-g \wedge t \leq s(p_l) < e(p) \wedge \neg(e'(p_l) \leq t) \wedge L_{p_l} \leq L_{a_i}} c_{p_l} +$

$\sum_{a_k \in A-h-k-a \wedge L_{a_k} < L_p \wedge (\neg(0 \leq R'(a_k)) \vee (R'(a_k) \leq t \wedge \neg(e'(a_k) \leq t)) \vee (R'(a_k) + \min_{a_k} < e(p)))} \left\lceil \frac{e(p) - t}{\min_{a_k}} \right\rceil \cdot c_{a_k}$

then Delay  $a_i$ ;

In Case 1:  $a_i$  is delayed either if there exists the possibility that the immediate execution of  $a_i$  may cause a P-g process  $p$  with less latitude to be delayed (as shown in Figure 20A); or, if there exists the

5 possibility that it may cause some A-h-k-a process  $a_j$  to be blocked for the duration of two processes  $a_i$  and  $p$  which both have greater latitude compared with  $a_j$ 's latitude (as shown in Figure 20B).

10 Case 2:

As shown in Figure 20C,

if  $\exists x, x \in P-g \vee x \in A-h-k-a$ :

$s'(x) < t \wedge \neg(e'(x) \leq t) \wedge x$  excludes  $a_i$  then Delay  $a_i$ ;

15 In Case 2:  $a_i$  is delayed because it is not allowed to preempt any process  $x$  that excludes  $a_i$ .

Case 3:

As shown in Figure 20D,

20 if  $\exists x, x \in P-g \vee x \in A-h-k-a$ :

$s'(x) < t \wedge \neg(e'(x) \leq t) \wedge L_x \leq L_{a_i}$   
then Delay  $a_i$ ;

In Case 3:  $a_i$  is delayed because it is not allowed to preempt any process  $x$  that has less or the

25 same latitude as  $a_i$ .

Case 4:

As shown in Figure 20E,

if  $\exists a_j, p, a_j \in A-h-k-a, p \in P-g$ :

30  $s'(p) \Rightarrow t \wedge s'(a_j) < t \wedge \neg(e'(a_j) \leq t)$

**THE UNIVERSITY OF CHICAGO**

Case 5:

10 if  $\exists p, p_1, p, p_1 \in P-g:$

$$\wedge (\text{not} \exists p_m, p_m \in P\text{-g: } t < s'(p_m) \wedge e_{pm} = s(p) \wedge L_{ai} < L_{pm})$$

15 C<sub>p1</sub>

then Delay  $a_i$ ;

25 Case 6:

if  $\exists x, a_j, x \in A\text{-h-k-a} \vee x \in P\text{-g}, a_j \in A\text{-h-k-a}$ :

 $\wedge x$  excludes  $a_j$ 

111

then Delay  $a_i$ .

In Case 6:  $a_i$  is delayed because it is not allowed to preempt any process  $x$  that excludes some other A-h-k-a process  $a_j$  which has a latitude that is less than both  $x$  and  $a_i$ 's latitude, because that may cause  $a_j$  to be blocked by the duration of more than one process with greater latitude.

Case 7:

10 As shown in Figure 20H,

if  $\exists p, p \in P-g$ :

$R_p \leq t \wedge (e'(p) \leq t) \wedge L_p \leq L_{a_i}$

$\wedge (s'(a_i) < t \wedge (a_i \text{ cannot be preempted by } p))$

$\wedge \text{not} \exists p_i, p_i \in P-g: s(p_i) \leq s(p) \wedge (e'(p_i) \leq t)$

15  $\wedge (s(p_i) \leq s(p) \wedge e(p) < e(p_i))$

then Delay  $a_i$ ;

In Case 7:  $a_i$  is delayed so that it can be preempted by a P-g process  $p$  that has a latitude that is less than or equal to  $a_i$ 's latitude, when  $a_i$  does not exclude  $p$  and does not exclude any A-h-k-a process with a latitude that is shorter than  $p$ 's latitude, and there does not exist any P-g process  $p_i$  that has not completed such that  $p_i$  is ordered before  $p$  and  $p$  does not preempt  $p_i$  in the pre-run-time schedule.

25

Select, among all processes  $a_i \in A-h-k-a$ , such that  $a_i$  has already arrived and not yet completed, and  $a_i$  is NOT Delayed, the process which has the least latitude. If more than one process is thus selected,

select among them the process that has the smallest index.

end;

5

return to Main Run-Time Scheduler.

The A-h-k-a Scheduler has the following properties:

10 Property 1. Each P-g process  $p$ 's execution can only be delayed by A-h-k-a processes that have less latitude than  $p$ 's latitude. A P-g process will never be delayed by any A-h-k-a process that has a greater or equal latitude.

15

Property 2. Any A-h-k-a process  $a_i$  cannot be blocked by more than one critical section belonging to A-h-k-a processes that have deadlines greater than  $a_i$ 's deadline.

20

Property 3. No deadlocks can ever occur.

Property 4. Each P-g process  $p$  will always be completed on or before  $e(p)$ , that is, the end of the time slot  
25 allocated to  $p$  in the pre-run-time schedule.

#### Example 11

Continuing with the set of processes in Examples 1-10 above, assume that A-h-k-a process  $a_2$  makes a  
30 request at time 99. Figure 6 is an illustration of this case, which is a possible run-time execution of the A-h-

k-a scheduling method together with the P-h-k and P-s-k methods described with regard to Example 11, and Figure 3, during the run-time phase.

Because  $s(\text{newp}_3) = 100 \Rightarrow t = 99 \wedge a_2$  excludes

5  $\text{newp}_3 \wedge L_{a_2} = d_{a_2} = 239 \Rightarrow L_{\text{newp}_3} = d_{\text{newp}_3} - r_{\text{newp}_3} = 115 -$   
 $100 = 15 \in e(p_3) - t = 114 - 99 = 15 < c_{a_2} + c_{p_3} + c_{a_0} +$   
 $c_{a_1} = 10 + 10 + 2 + 2 = 24$ , the conditions in Case 1 of  
the A-h-k-a Scheduler Subroutine are true, so  $a_2$  is  
delayed. If  $a_2$  is allowed to start at time 99, then  
10  $\text{newp}_3$  would miss its deadline. This is because  $a_2$   
excludes  $\text{newp}_3$ , once it starts, it cannot be preempted  
by  $\text{newp}_3$ , thus its execution will occupy a portion of  
the time frame reserved by  $\text{newp}_3$  in the pre-run-time  
schedule which only includes reserved processor capacity  
15 for itself and A-h-k-a processes with shorter deadlines,  
which are  $a_0$  and  $a_1$ .

Suppose that both  $a_0$  and  $a_1$  make requests at time  
130, then  $a_0$  will execute from time 130 to 132, and  $a_1$   
will execute from time 132 to 134, both using processor  
20 capacity that was reserved for them in  $p_6$ 's reserved  
time frame.

$a_2$  will be executed after  $\text{newp}_3$ ,  $p_6$ ,  $p_7$ ,  
since it's latitude  $L_{a_2} = d_{a_2} = 239$  is greater than the  
latitudes of  $\text{newp}_3$ ,  $p_6$ ,  $p_7$ .

25 Note that an A-h-k-a process  $a_i$  may be blocked  
from the start time to the end time of some critical  
section of a P-g process in the pre-run-time schedule.

One can easily determine the maximum time that  
any A-h-k-a process can be delayed by any P-g process  
30 with a simple inspection of the pre-run-time schedule.  
One can also easily make changes to a pre-run-time

schedule in order to avoid any excessive delay of an A-h-k-a process  $a_i$  that may be caused by preemptions of a P-g process's critical section by other P-g processes.

Above, A P-g process  $p_1$  has been allowed to be  
5 preempted by some other P-g process  $p_2$ , even if this may  
cause some A-h-k-a process  $a$  to be blocked by the  
duration of two critical sections belonging to two P-g  
processes  $p_1$  and  $p_2$  which both have latitudes that are  
greater than  $a$ 's latitude. This provides greater  
10 flexibility to the scheduling of P-g processes.  
However, it is easy to guarantee that any A-h-k-a  
process  $a$  cannot be blocked by the duration of two  
critical sections belonging to two P-g processes  $p_1$  and  
 $p_2$  which both have latitudes that are greater than  $a$ 's  
15 latitude. To guarantee this, all one needs to do is the  
following, for all pairs of P-g processes  $p_1$  and  $p_2$ , if  
 $p_1$  excludes some A-h-k-a process  $a$ , and  $L_{p(2)} \Rightarrow L_a$ , then  
add the exclusion relation  $p_1$  excludes  $p_2$ .

If the potential run-time overhead of the A-h-k-  
20 a Scheduler in the integration approach is compared with  
the overhead of methods that schedule all the tasks at  
run-time, the following may be noticed:

(a) With the integration approach, the number of  
processes  
25 that the A-h-k-a Scheduler needs to handle, should be  
very small. This is because, in most real-time systems,  
it has been observed that the bulk of the computation is  
performed by periodic processes, while the number of  
asynchronous processes with hard deadlines is usually  
30 very small. In addition a significant portion of the



asynchronous processes will be transformed into periodic processes when using this approach.

(b) The interarrival times of A-h-k-a processes that are not converted into new P-h-k processes are likely to be  
5 long.

(c) A significant portion of the parameters used by the A-h-k-a Scheduler to make scheduling decisions, are known before run-time, so one can pre-compute major portions of the conditions that are used for decision  
10 making, hence the amount of computation that needs to be performed for scheduling purposes at run-time can be minimized.

Thus the run-time overhead of the A-h-k-a Scheduler is believed to be far smaller than the  
15 overhead of methods that schedule all the tasks at run-time.

#### The Main Run-Time Scheduler

20 At run-time, the order of the execution of any pair of guaranteed periodic processes, i.e., P-h-k or P-s-k processes is kept consistent with the order of that pair of processes in the pre-run-time schedule.

A-s-u processes are scheduled at a lower  
25 priority than the A-h-k-a, P-h-k, P-s-k, and A-s-k processes. That is, A-s-u processes are executed only when there does not exist any process with known characteristics, i.e., A-h-k-a, P-h-k, P-s-k, or A-s-k process, that is ready for execution. If more than one  
30 A-s-u process are competing for execution, the process with the shortest deadline will be chosen for execution.

An A-s-u process is not allowed to have any precedence relation with any other process. An A-s-u process  $a_i$  is not allowed to have any exclusion relation of the form  $a_i$  excludes  $x$  where  $x$  is an A-h-k-a, P-h-k, P-s-k or A-s-k process. These restrictions are imposed in order to prevent A-s-u processes from causing any delay to processes with known characteristics; so that one can guarantee that all the hard deadlines will be satisfied, and also provide firm response time guarantees for all the processes with known characteristics.

For any P-g process  $p_1$  and any P-g process  $p_2$ , if  $s(p_1) < e(p_2) < e(p_1)$ , let  $p_2$ 's time slot be nested within  $p_1$ 's time slot in the pre-run-time schedule.

#### 15 Main-Run-Time-Scheduler Method:

At any time  $t$  if some process  $x$  has arrived at time  $t$ , or has completed at time  $t$ ,  
or if  $t$  is both the release time and start time in the pre-run-time schedule for some P-g process  $p$ , i.e.,  $t = R_p = s(p)$   
then execute the Main-Run-Time-Scheduler as follows:  
execute the A-h-k-a-Scheduler-Method described earlier;  
if some A-h-k-a process  $a_i$  is selected for execution at time  $t$  by the A-h-k-a Scheduler  
then execute  $a_i$   
else  
begin  
if  $\exists p, p \in P\text{-g}: s'(p) \leq t \wedge \neg(e'(p) \leq t)$   
 $\wedge \neg \exists p_i, p_i \in P\text{-g}: R_{p_i} \leq t \wedge \neg(e'(p_i) \leq t) \wedge$   
 $s(p) < s(p_i) \wedge e(p_i) < e(p)$

{there exists p that has started and has not  
 completed, and there does not exist any other  $p_i$  that is  
 ready and has not completed, such that  $P_i$ 's time slot is  
 nested within p's time slot in the pre-run-time  
 5 schedule}  
 then continue to execute p.  
 else  
 if  $\exists p, p \in P-g: R_p \leq t \wedge \neg(e'(p) \leq t)$   
 $\wedge \neg \exists p_i, p_i \in P-g: \neg(e'(p_i) \leq t) \wedge s(p_i) \leq$   
 10  $s(p) \wedge \neg(s(p_i) \leq s(p) \wedge e(p) < e(p_i))$   
 $\wedge \neg \exists p_j, p_j \in P-g: R_{p_j} \leq t \wedge \neg(e'p_j) \leq t) \wedge$   
 $s(p) < s(p_j) \wedge e(p_j) < e(p)$   
 {there exists p that is ready and has not  
 completed, and there does not exist any other  $p_i$  that  
 15 has not yet completed, such that  $p_i$  is ordered before p  
 in the pre-run-time schedule, and p's time slot is not  
 nested within  $p_i$ 's time slot in the pre-run-time  
 schedule, and there does not exist any other  $p_j$  that is  
 ready and has not completed, such that  $p_j$ 's time slot is  
 20 nested within p's time slot in the pre-run-time  
 schedule}  
 then execute p  
 else  
 if  $\exists a_i, a_i \in A-s-k: R_{a_i} \leq t \wedge \neg(e'(a_i) \leq t)$   
 25  $\wedge \neg \exists x: (s'(x) < t \wedge \neg(e'(x) \leq t) \wedge ((x$   
 excludes  $a_i)$   
 $\vee (\exists a_j, a_j \in A-s-k: s'(x) < t \wedge \neg(e'(x) \leq t) \wedge x$   
 excludes  $a_j$   
 $\wedge L_{a_j} < L_x$   
 30  $\wedge L_{a_j} < L_{a_i}))$

{there exists A-s-k process  $a_i$  that is ready and  
 has not completed, and there does not exist any other  
 process  $x$  such that  $x$  excludes  $a_i$  or  $x$  excludes some  
 process  $a_j$  such that  $a_j$  has a latitude that is less than  
 5 both  $x$ 's and  $a_i$ 's latitude, and  $x$  has started but not  
 completed}  
 then select among them, a process  $a_i$  that has the  
 shortest deadline;  
 if among such processes there are some that have  
 10 already started, then choose a process that has already  
 started; and execute  $a_i$ ;  
 else  
 if  $\exists a_i, a_i \in A-s-u: R_{ai} \leq t \wedge \neg(e'(a_i) \leq t)$   
 $\wedge \neg \exists x: (s'(x) < t \wedge \neg(e'(x) \leq t \wedge ((x$   
 15 excludes  $a_i)$   
 $\vee (\exists a_j, a_j \in \text{In } A-s-u: s'(x) < t \wedge \neg(e'(x)$   
 $\leq t) \wedge x \text{ excludes } a_j$   
 $\wedge L_{aj} < L_x$   
 $\wedge L_{aj} < L_{ai}$   
 20 {there exists A-s-u process  $a_i$  that is ready and  
 has not completed, and there does not exist any other  
 process  $x$  such that  $x$  excludes  $a_i$  or  $x$  excludes some  
 process  $a_j$  such that  $a_j$  has a latitude that is less than  
 both  $x$ 's and  $a_i$ 's latitude, and  $x$  has started but not  
 25 completed then select among them, a process  $a_i$  that has  
 the shortest deadline; if among such processes there are  
 some that have already started, then choose a process  
 that has already started; and execute  $a_i$ ;  
 end;

30

Example 12

Continuing with the set of processes in Examples 1-11 above, assume the following asynchronous process  
5  $a_{11}$  with a soft deadline and unknown characteristics (A-s-u process). ( $a_{11}$ 's characteristics are only known after its arrival.)

$a_{11}$ :  $c_{a_{11}} = 10$ ,  $d_{a_{11}} = 300$ .

10

Assume also that A-s-u process  $a_{11}$  makes a request at time 190; A-h-k-a process  $a_2$  makes a request at time 191; A-s-k process  $a_{10}$  makes a request at time 196; and A-h-k-a processes  $a_0$  and  $a_1$  make requests at  
15 time 200. Figure 7 illustrates an example of this case, in particular a run-time execution of the A-s-u process  $a_{11}$ , and the A-s-k process  $a_{10}$ , scheduled during main run-time scheduling together with the A-h-k-a processes  $a_0$ ,  $a_1$ ,  $a_2$  and the P-h-k and P-s-k processes described  
20 with regard to Figure 3, in Example 12, during the run-time phase.

At time 190  $a_{11}$  will be put into execution as there are no other processes that are ready for execution. At time 191  $a_2$  will be delayed because the  
25 conditions of Case 1 of the A-h-k-a Scheduler will be true. Note that  $a_2$  excludes  $newp_3$ , and  $L_{newp_3} < L_{a_2}$ ; if  $a_2$  is allowed to execute at time 191, it will cause  $newp_3$  to miss its deadline if  $a_0$  and  $a_1$  also preempt  $newp_3$ . At time 196  $a_{10}$  will preempt  $a_{11}$  as A-s-k processes are  
30 scheduled before A-s-u processes. At time 200  $a_0$  will

preempt  $a_{10}$  as A-h-k-a processes are scheduled before A-s-k processes.

$a_{11}$  will execute from time 190 to 196;  $a_{10}$  will execute from time 196 to 200;  $a_0$  will execute from time 200 to 202;  $a_1$  will execute from time 202 to 204;  $newp_3$  will execute from time 204 to 214. As  $L_{p_8} < L_{a_2}$ ,  $L_{p_5} < L_{a_2}$ , and  $L_{p_4} < L_{a_2}$ ;  $p_8$  will preempt  $a_2$  at time 220;  $p_8$ ,  $p_5$ ,  $p_4$  will execute from time 220 to 278;  $a_2$  will resume execution from time 278 to 282;  $a_{10}$  will resume execution from time 282 to 288;  $a_{11}$  will resume execution from time 288 to 292.

Note that each process may be completed earlier than the time indicated in the pre-run-time schedule, since the time that is reserved for each synchronous process with a shorter deadline in a guaranteed periodic process's time frame in the pre-run-time schedule will not be used by that asynchronous process if it does not arrive during that time frame.

### Example 13

In example 3, when using the procedure for converting a set of asynchronous processes into a set of new periodic processes,  $threshold(a_3)$  was set to 2.5, resulting in the A-h-k process  $a_3$  being converted into a new process  $newp_3$ .

Now assume that  $threshold(a_3)$  is set to an arbitrary large value, say 50, that would guarantee that  $a_3$  would not be converted into a periodic process.

In the case that A-h-k process  $a_3$  remains asynchronous, because the latitude of  $a_3$ ,  $L_{a_3} = d_{a_3} =$

114, as well as the latitudes of  $a_0$  and  $a_1$  are greater than the latitudes of the periodic processes  $p_4$ ,  $p_6$ ,  $p_7$  in meeting their respective deadlines, the worst-case computation times of  $p_4$ ,  $p_6$ ,  $p_7$  should be adjusted to  
 5 leave "room" for  $a_0$ ,  $a_1$  and  $a_3$ 's worst-case computation time as follows:

$$C'_{p_4} = C_{p_4}C_{a_0} + C_{a_1} + C_{a_3} = 26 + 2 + 2 + 10 = 40$$

$$C'_{p_6} = C_{p_6}C_{a_0} + C_{a_1} + C_{a_3} = 26 + 2 + 2 + 10 = 40$$

$$C'_{p_7} = C_{p_7}C_{a_0} + C_{a_1} + C_{a_3} = 16 + 2 + 2 + 10 = 30$$

10           The pre-run-time scheduler will first construct the feasible pre-run-time schedule shown in Figure 8 for the set of P-g processes  $p_4$ ,  $p_5$ ,  $p_6$ ,  $p_7$ . Then the simulation procedure for determining the worst-case response time of an A-h-k-a process can be used to  
 15 determine  $a_0$ ,  $a_2$ ,  $a_3$ ,  $a_9$ 's worst-case response time. A-h-k-a process  $a_3$ 's worst-case response time happens when  $a_3$  arrives at time 7, while  $a_2$  which excludes  $a_3$  arrived one time unit before  $a_3$  at time 6. If  $a_0$ ,  $a_1$  which have less latitude than  $a_3$  in meeting their respective  
 20 deadlines, arrive at the same time 6,  $a_0$  will preempt  $a_2$  and execute from time 7 to time 9,  $a_1$  will execute from time 9 to time 11,  $a_2$  will continue its execution from time 11 to time 20,  $p_8$  will first execute from time 20 to time 30; at time 30  $p_8$  will be preempted by  $p_5$  which  
 25 executes from time 30 to time 46; at time 46  $p_8$  will continue its execution from time 46 to time 52. At time 52, because  $L_{a_3} = D_{a_3} = 114 < L_{p_4} = d_{p_4} - r_{p_4} = 200 - 0 = 200$ ,  $a_3$  will execute from time 52 to time 62. A-h-k-a process  $a_3$ 's worst-case response time  $RE_{a_3} = \{RE(a_3, t_8)\}$   
 30  $= RE(a_3, 7) = e'(a_3) - R_{a_3} = 62 - 7 = 45 \leq d_{a_3} = 114$ .  
 Similarly, one can verify that the worst-case response

times of all the other asynchronous processes are all less than or equal to their respective deadlines, as shown in Figure 8.

The embodiments described herein are  
5 advantageous methods of the integration approach compared with methods that perform all scheduling activities at run-time. It should be noted that existing methods or protocols that perform all scheduling activities at run-time are not able to  
10 guarantee the schedulability of the set of processes given in these examples.

There are many reasons for this, including:

1. Prior art run-time scheduling methods are not capable of finding optimal schedules involving critical  
15 sections, except for the simplest problem instances, because not enough time is available to the scheduler at run-time.
2. Prior art run-time scheduling methods cannot handle precedence  
20 constraints, release time and exclusion constraints simultaneously in an efficient way.
3. Current run-time scheduling methods are unable to take full advantage of the knowledge about processes characteristics that is available before run-time. For  
25 example, no prior art run-time scheduling method can completely avoid blocking of a periodic process with less latitude by a asynchronous process with greater latitude, which the integration method described herein is capable of doing, as shown in the examples above. As  
30 another example, when determining the worst-case response times of asynchronous processes, no prior art



run-time scheduling method can completely avoid making the overly pessimistic assumption that, for each process, all the periodic processes with shorter deadlines can arrive at the same time to delay that  
5 process. In contrast, the integration method in accordance with the present invention can avoid making such an overly pessimistic assumption, as shown in the examples where a simulation procedure can obtain tighter worst-case response times for asynchronous processes, by  
10 taking advantage of the knowledge of the positions of the periodic processes in the pre-run-time schedule.

If the potential run-time overhead of the Main Run-Time Scheduler in the integration approach is compared with  
15 the overhead of methods that schedule all the tasks at run-time, the following will be noticed:

(a) The Main Run-Time Scheduler is much simpler, and the amount of computation needed for scheduling purposes is much smaller, compared with most methods that schedule  
20 all the tasks at run-time. This is because most of the important scheduling decisions have already been made before run-time. In particular, the relative ordering of P-h-k and P-s-k processes that usually form the bulk of the computation in most real-time applications, was  
25 determined before run-time when the pre-run-time schedule was computed.

(b) Since at run-time, the execution order of P-h-k and P-s-k processes is the same as the relative ordering of those processes in the pre-run-time schedule, one would  
30 know exactly which guaranteed periodic process may preempt which other guaranteed periodic process at run-

time. Thus one can use this information to minimize the  
amount of context switching. Thus it is believed that  
the run-time overhead of the Main Run-Time Scheduler  
should be by far smaller than the overhead of methods  
5 that schedule all the tasks at run-time.

## Using the Present Invention With Multiple Processors

The methods described above can be used for scheduling processes with exclusion relations, precedence relations, and offset constraints, release time, worst-case computation time, deadline constraints, on more than one processor.

There are many possible ways that would allow one to use the methods with more than one processor. The following is just one possible set of changes to the procedures described earlier that would allow one to use the methods for scheduling processes on more than one processor. The use of the methods with multiple processors is illustrated in Example 14.

As stated earlier, instead of using a single processor method such as the method in [XuPa90] in the procedure for constructing a feasible pre-run-time schedule for a given set of periodic processes with offsets, one should use a multiple processor scheduling method, such as the method in [Xu93] in that procedure.

One simple strategy, that will be used in the embodiment described below, is to set the release time  $r_{p_i}$  of every periodic process  $p_i$  to be equal to the beginning time of its time slot in the pre-run-time schedule, i.e.,  $r_{p_i} = s(p_i)$ . This ensures that every periodic process' actual execution will not start earlier than the beginning time of its time slot, i.e.,  $r_{p_i} \leq s'(p_i)$ . This could prevent multiple processor anomalies that could be caused by the following situation. Some processes end earlier, and a first process that excludes a second process with a relatively short deadline is executed earlier, resulting in the first process' execution combining with the execution of a third process on another processor that also excludes the second process to increase the time interval in which the second process is blocked from execution. It is not difficult to design alternative methods that would allow each periodic process to start execution at a time earlier than the beginning of its time slot, while preventing such anomalies. Likewise, in the design of the Multiple Processor A-h-k-a Scheduler described below, there are many possible alternative strategies that would allow some more flexibility in scheduling the processes, however, this disclosure would be of too great length if every possible improvement is described.

The Multiple Processor A-h-k-a Scheduler and Multiple Processor Main-Run-Time Scheduler can be designed as follows.

Each time the Multiple Processor Run-Time Scheduler is executed, it will first  
 5 try to schedule A-h-k-a processes according to the procedure below.

For any A-h-k-a process  $a_i$  and any P-g process  $p_l$ , it will be said that  
 “ $a_i$  cannot be preempted by  $p_l$ ”, if the following conditions should hold:

$(a_i \text{ excludes } p_l) \vee (\exists a_j, a_j \in \text{A-h-k-a: } L_{a_j} < L_{p_l} \wedge a_i \text{ excludes } a_j)$

For any pair of P-g processes  $p_1$  and  $p_2$ , it will be said that  
 10 “the time slot of  $p_1$  overlaps with the time slot of  $p_2$ ”, if the following conditions should hold:

$(s(p_1) \leq s(p_2) < e(p_1)) \vee (s(p_2) \leq s(p_1) < e(p_2))$

The Multiple Processor A-h-k-a Scheduler Method functions as part of the Multiple  
 15 Processor Main-Run-Time Scheduler to be described below.

Multiple Processor A-h-k-a-Scheduler Method:

At any time  $t$ :

20 if some A-h-k-a process  $a_i$  has arrived at time  $t$ ,  
 or if some process  $x_i$  completes its computation at time  $t$   
 or if  $t$  is both the release time and start time in the pre-run-time schedule  
 for some P-g process  $p$ , i.e.,  $t = R_p = s(p)$

then

25 for every processor  $k$ :

begin

for each A-h-k-a process  $a_i$  that has already arrived and not yet completed,  
 i.e.,  $R'(a_i) \leq t \wedge \neg(e'(a_i) \leq t)$ , check if  $a_i$   
 satisfies the following conditions,

30

Case 1: if on any processor  $k$ ,

$\exists p, p \in \text{P-g:}$

$s'(p) \geq t \wedge (a_i \text{ cannot\_be\_preempted\_by } p) \wedge L_{a_i} \geq L_p$   
 $\wedge (\nexists p_m, p_m \in P\text{-g: } t < s'(p_m) \wedge e(p_m) \leq s(p) \wedge L_{a_i} < L_{p_m} \wedge (\nexists p_1 \in P\text{-g:}$   
 $p_1 \text{ excludes } a_i \wedge (p_1 \text{ 's time slot overlaps with } p_m \text{ 's time slot}))$   
 $\wedge (\text{there does not exists any interval } [t, t_2] \text{ on processor } k, \text{ such that:}$

- 5  $t < t_2 \leq s(p)$
- $\wedge (\nexists p_l, p_l \in P\text{-g: any portion of } p_l \text{ 's time slot is mapped to any portion of the}$   
interval  $[t, t_2]$  on the time axis corresponding to processor  $k$  in the  
pre-run-time schedule)
- $\wedge (\nexists p_j, p_m \in P\text{-g: } p_j \text{ excludes } a_i \wedge (p_j \text{ 's time slot overlaps with } [t, t_2]))$
- 10  $\wedge (t_2 - t) \geq c_{a_i} +$
- $\sum_{a_k \in A\text{-h-k-a} \wedge L_{a_k} < L_p \wedge (\neg(0 \leq R'(a_k)) \vee (R'(a_k) \leq t \wedge \neg(e'(a_k) \leq t)) \vee (R'(a_k) + \min_{a_k} < s(p))) \lceil \frac{t_2 - t}{\min_{a_k}} \rceil \cdot c_{a_k}$
- then Delay  $a_i$ ;

- 15 In Case 1:  $a_i$  is delayed either if there exists the possibility that the immediate execution of  $a_i$  may cause a P-g process  $p$  with less latitude to be delayed (as shown in Figure 20A); or, if there exists the possibility that it may cause some A-h-k-a process  $a_j$  to be blocked for the duration of two processes  $a_i$  and  $p$  which both have greater latitude compared with  $a_j$ 's latitude (as shown in Figure 20B).

- 20 Case 2: if on any processor  $k$ ,
- $\exists x, x \in P\text{-g} \vee x \in A\text{-h-k-a:}$
- $s'(x) < t \wedge \neg(e'(x) \leq t) \wedge x \text{ excludes } a_i$
- then Delay  $a_i$ ;

- 25 In Case 2:  $a_i$  is delayed because it is not allowed to start its execution if there exists any process  $x$  that excludes  $a_i$  that has started but not yet completed (as shown in Figure 20C).

- Case 3: if on processor  $k$
- 30  $\exists x, x \in P\text{-g} \vee x \in A\text{-h-k-a:}$
- $s'(x) < t \wedge \neg(e'(x) \leq t) \wedge L_x \leq L_{a_i}$
- then  $a_i$  is ineligible to execute at time  $t$  on processor  $k$ ;

In Case 3:  $a_i$  is ineligible to execute at time  $t$  on processor  $k$  because it is not allowed to preempt any process  $x$  that has less or the same latitude as  $a_i$  (as shown in Figure 20D).

5

Case 4: if on processor  $k$ ,

$\exists a_j, p, a_j \in A\text{-h-k-a}, p \in P\text{-g}$ :

$s'(p) \geq t \wedge s'(a_j) < t \wedge \neg(e'(a_j) \leq t)$

$\wedge a_j \text{ excludes } p \wedge L_p \leq L_{a_i}$

10 then  $a_i$  is ineligible to execute at time  $t$  on processor  $k$

In Case 4:  $a_i$  is is ineligible to execute at time  $t$  on processor  $k$  because it is not allowed to preempt any A-h-k-a process  $a_j$  which excludes a P-g process  $p$  with less or equal latitude compared with  $a_i$ 's latitude. (as shown in Figure 20E)

15

Case 5: on processor  $k$ ,

if  $\exists x, a_j, x \in A\text{-h-k-a} \forall x \in P\text{-g}$ ,

$a_j \in A\text{-h-k-a}$ :

$s'(x) < t \wedge \neg(e'(x) \leq t)$

20  $\wedge x \text{ excludes } a_j$

$\wedge L_{a_j} < L_x \wedge L_{a_j} < L_{a_i}$

then  $a_i$  is ineligible to execute at time  $t$  on processor  $k$ .

In Case 5:  $a_i$  is ineligible to execute at time  $t$  on processor  $k$  because it is not allowed to preempt any process  $x$  that excludes some other A-h-k-a process  $a_j$  which has a latitude that is less than both  $x$  and  $a_i$ 's latitude, because that may cause  $a_j$  to be blocked by the duration of more than one process with greater latitude (as shown in Figure 20G).

30 Case 6: if on processor  $k$

$\exists p, p \in P\text{-g}$ :

$R_p \leq t \wedge \neg(e'(p) \leq t) \wedge L_p \leq L_{a_i}$

$\wedge \neg(s'(a_i) < t \wedge (a_i \text{ cannot\_be\_preempted\_by } p))$

$\wedge \nexists p_i, p_i \in P\text{-g}: s(p_i) \leq s(p) \wedge \neg(e'(p_i) \leq t)$

$\wedge \neg(s(p_i) \leq s(p) \wedge e(p) < e(p_i))$

then  $a_i$  is ineligible to execute at time  $t$  on processor  $k$ ;

5

In Case 6:  $a_i$  is ineligible to execute at time  $t$  on processor  $k$  so that it can be preempted by a P-g process  $p$  that has a latitude that is less than or equal to  $a_i$ 's latitude, when  $a_i$  does not exclude  $p$  and does not exclude any A-h-k-a process with a latitude that is shorter than  $p$ 's latitude (as shown in Figure 20H).

10 end;

For each processor  $k$ , select, among all processes  $a_i \in A\text{-h-k-a}$ , such that  $a_i$  has already arrived and not yet completed, and  $a_i$  is NOT Delayed, and  $a_i$  is NOT ineligible to execute at time  $t$  on processor  $k$ , the process which has the shortest deadline and execute

15 that process on processor  $k$ . If more than one process is thus selected, select among them the process that has the smallest index.

end;

20 return to Multiple Processor Main Run-Time Scheduler;

Multiple Processor Main-Run-Time-Scheduler Method:

25

At any time  $t$ :

if some process  $x$  has arrived at time  $t$ , or has completed at time  $t$ ,

or if  $t$  is both the release time and start time in the pre-run-time schedule

for some P-g process  $p$ , i.e.,  $t = R_p = s(p)$

30 then execute the Multiple Processor Main-Run-Time-Scheduler as follows:

execute the Multiple Processor A-h-k-a-Scheduler;

For each processor  $k$ , if some A-h-k-a process  $a_i$  was selected for execution at time  $t$  on processor  $k$  by the A-h-k-a Scheduler then execute  $a_i$

5 else begin

if on processor  $k$ ,

$\exists p, p \in P\text{-g}: s'(p) \leq t \wedge \neg(e'(p) \leq t)$

(there exists  $p$  that has started and has not completed)

then continue to execute  $p$ .

10 else

if on processor  $k$ ,

$\exists p, p \in P\text{-g}: R_p \leq t \wedge \neg(e'(p) \leq t)$

(there exists  $p$  that is ready and has not completed)

then execute  $p$

15 else

if  $\exists a_i, a_i \in A\text{-s-k}: R_{a_i} \leq t \wedge \neg(e'(a_i) \leq t)$

$\wedge \nexists x : (s'(x) < t \wedge \neg(e'(x) \leq t) \wedge ((x \text{ excludes } a_i)$

$\vee (\exists a_j, a_j \in A\text{-s-k}: s'(x) < t \wedge \neg(e'(x) \leq t) \wedge x \text{ excludes } a_j$

$\wedge L_{a_j} < L_x \wedge L_{a_j} < L_{a_i}))$

20

(there exists A-s-k process  $a_i$  that is ready and has not completed,

and there does not exist any other process  $x$  such that  $x$  excludes

$a_i$  or  $x$  excludes some process  $a_j$  such that  $a_j$  has a latitude that is

less than both  $x$ 's and  $a_i$ 's latitude, and  $x$  has started but not completed)

then select among them, a process  $a_i$  that has the shortest

25

deadline; if among such processes there are some that have already

started, then choose a process that has already started; and execute

$a_i$ ;

else

if  $\exists a_i, a_i \in A\text{-s-u}: R_{a_i} \leq t \wedge \neg(e'(a_i) \leq t)$

30

$\wedge \nexists x : (s'(x) < t \wedge \neg(e'(x) \leq t) \wedge ((x \text{ excludes } a_i)$

$\vee (\exists a_j, a_j \in A\text{-s-u}: s'(x) < t \wedge \neg(e'(x) \leq t) \wedge x \text{ excludes } a_j$



$\wedge L_{a_j} < L_x \wedge L_{a_j} < L_{a_i}))$

(there exists A-s-u process  $a_i$  that is ready and has not completed,  
and there does not exist any other process  $x$  such that  $x$  excludes  
 $a_i$  or  $x$  excludes some process  $a_j$  such that  $a_j$  has a latitude that  
is less than both  $x$ 's and  $a_i$ 's latitude, and  $x$  has started but not  
completed)

then select among them, a process  $a_i$  that has the shortest deadline;  
if among such processes there are some that have already started,  
then choose a process that has already started; and execute  $a_i$ ;

end;

The multiple processor simulation method for determining the worst-case response time of A-h-k-a processes can be designed as follows:

Multiple processor method for computing the worst-case response time of an A-h-k-a process  $a_i$  corresponding to a feasible pre-run-time schedule of guaranteed periodic processes consisting of an initial part of the pre-run-time schedule  $S_0(t_0)$ , in the interval  $[0, t_0]$ ; and a repeating part of the pre-run-time schedule  $S_{LCM}(prd_{LCM})$ , in the interval  $[t_0, t_0 + prd_{LCM}]$ :

10 fail:= false;  
for  $t_s := 0$  to  $t_0 + prd_{LCM} - 1$  do  
begin  
For each A-h-k-a process  $a_b$ , such that  $a_b \in \text{A-h-k-a} \wedge L_{a_b} > L_{a_i} \wedge \exists x_k, x_k \in \text{A-h-k-a} \vee x_k \in \text{P-g: } a_b \text{ excludes } x_k \wedge L_{x_k} \leq L_{a_i}$ , do the following:  
15 let  $a_b$  arrive at time  $t_s - 1$ , and use the Multiple Processor A-h-k-a Scheduler and Multiple Processor Main Run-Time Scheduler to schedule  $a_b$  and  $a_i$  to obtain a response time of  $a_i$ , called  $RE_1(a_i, t_s, a_b)$ , corresponding to each  $a_b$  according to the assumptions (1)-(6) below, with the additional assumption that  $a_b$  can always start its execution at time  $t_s - 1$  (including when  $t_s = 0$ )  
20 except if at time  $t_s - 1$  there exists some periodic process  $p$ , such that  $t_s < s(p)$  and  $(a_b \text{ cannot\_be\_preempted\_by } p) \wedge L_{a_b} \geq L_p$ , and executing  $a_b$  at time  $t_s - 1$  may cause  $p$  to miss its deadline, that is,  $a_b$  is delayed by the conditions in Case 1 of the Multiple Processor A-h-k-a Scheduler. At time  $t = t_s - 1$ , assume that the condition " $s'(p) \geq t$ " is true for every periodic process  $p$  such that  $t_s < s(p)$   
25 when checking whether  $a_b$  should be delayed by the conditions of Case 1 of the Multiple Processor A-h-k-a Scheduler. When computing  $RE_1(a_i, t_s, a_b)$ , if  $\exists p_l, p_l \in \text{P-g} \wedge s(p_l) \leq t_s < e(p_l)$ , then assume that  $p_l$  has already completed at time  $t_s - 1$ , that is, this instance of  $p_l$  will not have any effect on  $a_b$ ,  $a_i$ , or any other process in the system.

30

(Let each A-h-k-a process  $a_b$  that can possibly block  $a_i$ , arrive at time  $t_s - 1$  and determine which one among them will block  $a_i$  for the greatest amount of time.)

After obtaining  $RE_1(a_i, t_s, a_b)$  for every such  $a_b$ , use the Multiple Processor A-h-k-a Scheduler and Multiple Processor Main Run-Time Scheduler to schedule  $a_i$  to obtain another response time of  $a_i$ , called  $RE_2(a_i, t_s)$ , according to the assumptions (1)-(6) below, with the additional assumption that no A-h-k-a process  $a_b$ , such that  $a_b \in \text{A-h-k-a} \wedge L_{a_b} > L_{a_i} \wedge \exists x_k, x_k \in \text{A-h-k-a} \forall x_k \in \text{P-g}: a_b \text{ excludes } x_k \wedge L_{x_k} \leq L_{a_i}$  had ever arrived. When computing  $RE_2(a_i, t_s)$ , if  $\exists p_l, p_l \in \text{P-g} \wedge s(p_l) \leq t_s < e(p_l)$ , then assume that  $s'(p_l) = s(p_l) \wedge e'(p_l) = e(p_l)$ ; {i.e.,  $p_l$  started at the beginning of its time slot and will complete its computation at the end of its time slot in the pre-run-time schedule that was computed using adjusted computation times.

(1)  $a_i$  arrives in the system at time  $t_s$ .

(2) Unless  $a_i$  is delayed because of the conditions in Case 1 of the Multiple Processor A-h-k-a Scheduler, (see (3) below), let the A-h-k-a processes  $a_j$  in the set  $\{a_j | L_{a_j} \leq L_{a_i} \wedge a_j \text{ excludes } a_i\}$  arrive one by one in a serial sequence such that each process in the set arrives exactly at the same instant that the process before it in the sequence has just completed, with the first process in the sequence arriving at the time that  $a_b$  has just completed if  $a_b$  is able to block  $a_i$ , and at the earliest time  $t$ ,  $t_s \leq t$  that it can be executed, if  $a_b$  does not block  $a_i$ ; let every other A-h-k-a process  $a_j$ , such that  $L_{a_j} \leq L_{a_i}$  arrive at the time that the last process in the above serial sequence has completed; or at the following instants in time:  $R_{a_j}(k) = t_s + k * \min_{a_j}$ ,  $k = 0, 1, 2, \dots, \lfloor \frac{d_{a_i}}{\min_{a_j}} \rfloor$ , if no such processes in the above set exist, and be scheduled before  $a_i$  whenever  $a_i$  and  $a_j$  have both arrived and  $a_i$  has not yet started. If  $a_j$  is delayed by any of the conditions in Cases 1-6 of the Multiple Processor A-h-k-a Scheduler then let  $a_i$  be delayed by the same amount of time.

For each such  $a_j$ 's subsequent arrival times, use the arrival times:  $R_{a_j}(k) = t_s + k * \min_{a_j}$ , and whenever there is more than one such process arriving at a time that any such process is executing or has arrived but not completed, modify their arrival times as described above.

5

( All A-h-k-a processes that have shorter deadlines and that exclude  $a_i$  arrive in a serial sequence that maximizes the time that  $a_i$  is excluded, all other A-h-k-a processes whose deadlines are shorter or equal to  $a_i$ 's deadline arrive at the end of that sequence if any, and are put into execution before  $a_i$ .

10

(3) Whenever the conditions in Case 1 of the Multiple Processor A-h-k-a Scheduler become true for  $a_i$  and some P-h-k process  $p$  at some time  $t$ , i.e.: if on any processor  $k$ ,  $\exists p, p \in P\text{-g}$ :

$$s'(p) \geq t \wedge (a_i \text{ cannot be preempted by } p) \wedge L_{a_i} \geq L_p$$

$$\wedge (\nexists p_m, p_m \in P\text{-g}: t < s'(p_m) \wedge e(p_m) \leq s(p) \wedge L_{a_i} < L_{p_m} \wedge (\nexists p_1 \in P\text{-g}: p_1 \text{ excludes } a_i \wedge (p_1 \text{'s time slot overlaps with } p_m \text{'s time slot}))$$

15

$$\wedge (\text{there does not exist any idle interval } [t, t_2] \text{ on some processor } k, \text{ such that:}$$

$$t < t_2 \leq s(p)$$

$$\wedge (\nexists p_j, p_m \in P\text{-g}: p_j \text{ excludes } a_i \wedge (p_j \text{'s time slot overlaps with } [t, t_2]))$$

$$\wedge (t_2 - t) \geq c_{a_i} +$$

20

$$\sum_{a_k \in A\text{-h-k-a} \wedge L_{a_k} < L_p \wedge ((\neg(0 \leq R'(a_k)) \vee (R'(a_k) \leq t \wedge \neg(e'(a_k) \leq t)) \vee (R'(a_k) + \min_{a_k} < s(p))) \left\lceil \frac{t_2 - t}{\min_{a_k}} \right\rceil \cdot c_{a_k}$$

let  $t$  also be the earliest time that the conditions in Case 1 become true for that instance of  $p$  and  $a_i$ ,

25

then for every other A-h-k-a process  $a_j$  that belongs to the following set:

$$\{a_j | a_j \in A\text{-h-k-a} \wedge j \neq i \wedge L_{a_j} \leq L_{a_i} \wedge$$

$$((\neg(0 \leq R'(a_j)) \vee (R'(a_j) \leq t \wedge \neg(e'(a_j) \leq t)) \vee (R'(a_j) + \min_{a_j} < s(p)))$$

$$\vee (s'(a_b) \leq t_s \wedge t - 1 \leq e'(a_b) \wedge s'(a_j) < t))\}$$

30

( $a_j$  has a deadline that is shorter than or equal to  $a_i$ 's deadline and, either  $a_j$  has never arrived; or  $a_j$  has arrived but has not completed its computation; or  $a_j$  may arrive again before time  $s(p)$ ; or at time  $t - 1$   $a_i$  was blocked by  $a_b$  and  $a_j$  started its execution before  $t$ ),

"delay"  $a_j$ 's arrival time to occur at the following time: let the A-h-k-a processes  $a_j$  in the set  $\{a_j | L_{a_j} \leq L_{a_i} \wedge a_j \text{ excludes } a_i\}$  arrive one by one in a serial sequence such that each process in the set arrives exactly at the same instant that the process before it in the sequence has just completed, with the first process in the sequence arriving at the time that  $p$  has just completed; let every other A-h-k-a process  $a_j$ , such that  $L_{a_j} \leq L_{a_i}$  arrive at the time that the last process in the above serial sequence has completed; or at the completion time of  $p$ ,  $e(p)'$ , if no such processes in the above set exist, and be scheduled before  $a_i$  whenever  $a_i$  and  $a_j$  have both arrived and  $a_i$  has not yet started. If  $a_j$  is delayed by any of the conditions in Cases 1-6 of the Multiple Processor A-h-k-a Scheduler, then let  $a_i$  be delayed by the same amount of time. If at time  $s(p)$  the conditions in Case 1 of the Multiple Processor A-h-k-a Scheduler continue to be true for some other periodic process  $p'$ , then  $a_i$  should be delayed again, and the A-h-k processes  $a_j'$  that satisfy the conditions of the formula for  $p'$  should also be delayed again in similar manner. For each such  $a_j$ , let only a single instance of  $a_j$  arrive at the above times, even if originally there could be several instances of a same process  $a_j$  that satisfy the conditions above.

For each such  $a_j$ 's subsequent arrival times after  $s(p)$ , use the same arrival times that were determined in (2), i.e.,  $a_j$ 's subsequent arrival times after  $s(p)$  will be:  $R_{a_j}(k) = t_s + k * \min_{a_j}$  such that  $R_{a_j}(k) > s(p)$ , and whenever there is more than one such process arriving at a time that any such process is executing or has arrived but not completed, modify their arrival times as described above.

(If at time  $t$  there exists more than one process  $p$  for which the conditions of Case 1 are true for  $p$  and  $a_i$ , then let the above apply to the process  $p$  among them that has the latest  $s(p)$  time in the pre-run-time schedule.)

( if  $a_i$  is delayed due to the conditions in Case 1, then an A-h-k-a process  $a_j$  could delay  $a_i$  by a maximum amount by arriving at the above described times.)

(4) If the end of the current instance of the repeating part of the pre-run-time schedule is reached, continue at the beginning of the next instance of the repeating part of the pre-run-time schedule.

(5) If  $a_i$ 's deadline  $d_{a_i}$  is reached but  $a_i$  has not yet completed its computation, then set  $fail := true$  and exit from the procedure.

(6) If  $a_i$ 's computation is completed before its deadline  $d_{a_i}$ , then record the completion time of  $a_i$  as the response time of  $a_i$  (either  $RE_1(a_i, t_s, a_b)$  for the current  $a_b$ , or  $RE_2(a_i, t_s)$  when no such  $a_b$  is assumed to have arrived at time  $t_s - 1$ ).

After the response time  $RE_1(a_i, t_s, a_b)$  corresponding to every  $a_b$  that may block  $a_i$  has been computed, and the response time  $RE_2(a_i, t_s)$  that assumed that no such  $a_b$  had arrived and blocked  $a_i$  has also been computed, set the response time of  $a_i$  with arrival time  $t_s$ ,  $RE(a_i, t_s)$  to be equal to the greatest among them, i.e.,  $RE(a_i, t_s) = \max\{RE_1(a_i, t_s, a_b), RE_2(a_i, t_s)\}$ , and exit from the current iteration of the loop for this value of  $t_s$ , and start the next iteration for  $t_s := t_s + 1$ .

end;

if not fail then

$RE_{a_i} := \max\{RE(a_i, t_s) \mid t_s = 0, 1, \dots, LCM - 1\};$

Assuming that the same `adjusted_capacity` function is used for adjusting the computation times of periodic processes, the Multiple Processor A-h-k-a Scheduler also has the following properties, similar to the properties of the version of the A-h-k-a Scheduler described earlier :

5

Property 1. Each P-g process  $p$ 's execution can only be delayed by A-h-k-a processes that have less latitude than  $p$ 's latitude. A P-g process will never be delayed by any A-h-k-a process that has a greater or equal latitude.

10

Property 2. Any A-h-k-a process  $a_i$  cannot be blocked by more than one critical section belonging to A-h-k-a processes that have deadlines greater than  $a_i$ 's deadline.

Property 3. No deadlocks can ever occur.

15

Property 4. Each P-g process  $p$  will always be completed on or before  $e(p)$ , that is, the end of the time slot allocated to  $p$  in the pre-run-time schedule.

#### Example 14

20 Suppose that a multiprocessor system consists of two processors, two asynchronous processes with hard deadlines and known characteristics (A-h-k processes):  $a_A$  and  $a_E$ ; and 3 periodic processes with hard deadlines and known characteristics (P-h-k processes)  $p_B$ ,  $p_C$ , and  $p_D$  as follows.

$a_A$ :  $c_{a_A} = 2, d_{a_A} = 8, \min_{a_A} = 8$ ;

25  $a_E$ :  $c_{a_E} = 2, d_{a_1} = 14, \min_{a_1} = 1,000$ ;

$p_B$ :  $r_{p_B} = 1, c_{p_B} = 3, d_{p_B} = 4, prd_{p_B} = 6, 0 \leq o_{p_B} \leq 3$ ;

$p_C$ :  $r_{p_C} = 0, c_{p_C} = 1, d_{p_C} = 2, prd_{p_C} = 4; o_{p_D} = 0$ ;

$p_D$ :  $r_{p_D} = 0, c_{p_D} = 1, d_{p_D} = 1, prd_{p_D} = 3, 0 \leq o_{p_D} \leq 4$ ;

It is assumed that the application requires that  $o_{p_C} = 0, 0 \leq o_{p_B} \leq 3$ , and  
 30  $0 \leq o_{p_D} \leq 4$ , and asynchronous process  $a_A$  may make its first request at time 0. It is also assumed that the application requires the following relations be satisfied:  $a_A$  excludes  $p_D$ ,  $p_D$  excludes  $a_A$ ,  $a_A$  excludes  $p_B$ ,  $p_B$  excludes  $a_A$ ,  $p_C$  excludes  $p_D$ ,  $p_D$  excludes  $p_C$ ,

$a_E$  excludes  $p_B$ ,  $p_B$  excludes  $a_E$ ;

Suppose that when converting  $a_A$  to a periodic process, when determining  $d_{newp_A}$ :

$$d_{newp_A} = c_{a_A} + conversion\_room(a_A);$$

5

the following formula was used in the `conversion_room` function:

$$conversion\_room(a_A) =$$

10

$$\left\lceil \frac{\sum_{p_j \in (S_P \cup S_p) \wedge d_{p_j} \leq d_{x_i}} \left\lceil \frac{d_{x_i}}{prd_{p_j}} \right\rceil * c_{p_j}}{m} \right\rceil + \sum_{a_j \in S_a \wedge d_{a_j} \leq d_{x_i} \wedge i \neq j} \left\lceil \frac{d_{x_i}}{min_{a_j}} \right\rceil * c_{a_j}$$

where  $m$  is the number of processors.

15

Then in the procedure for converting A-h-k processes into periodic processes, prior to entering the while loop,  $d_{newp_A} = c_A + (c_B + c_C + c_D)/m = 2 + \lceil (2 + 1 + 1)/2 \rceil = 4$ . After the second iteration of the while loop,  $d_{newp_A} = c_A + \lceil (c_C + c_D)/m \rceil = 2 + \lceil (1 + 1)/2 \rceil = 3$ .  $prd_{newp_A} = (d_A - d_{newp_A} + 1) = 8 - 3 + 1 = 6$ ;  $0 \leq o_{newp_A} \leq prd_{newp_A} - 1 = 6 - 1 = 5$ .

20

Suppose further that the system designer wanted to increase the chances of  $a_A$  being converted into a periodic process, so the  $threshold(a_A)$  was assigned a low value of 0.5.

$$RPC_{newp_A} = c_{newp_A}/prd_{newp_A} = 2/6 = 0.33.$$

$$RPC_{a_j} = c_{a_A}/min_{a_j} = 2/8 = 0.25.$$

As  $threshold(a_A) * RPC_{newp_A} = 0.5 * 0.33 \leq RPC_{a_A} = 0.25$ , the procedure will convert  $a_A$  into a periodic process  $newp_A$ .

25

Suppose also that the system designer did not want  $a_E$  to be converted into a periodic process, so  $threshold(a_E)$  was assigned a high value of 50. The procedure will not convert  $a_E$  into a periodic process.

30

If the method in [Xu93] was used by the procedure for constructing a feasible pre-run-time schedule for a given pre-run-time schedule with offsets, it will find the feasible schedule for  $newp_A$ ,  $p_B$ ,  $p_C$ , and  $p_D$ , on two processors illustrated in Fig. 20, when the offsets are set to the following values:  $o_{newp_A} = 2$ ,  $o_{p_B} = 3$ , and  $o_{p_D} = 1$ . The feasible schedule consists of a non-repeating initial schedule  $S_0(1)$  for the interval



[0,1], and a repeating schedule  $S_{LCM}(12)$  that is of length equal to the Least Common Multiple of the periods of  $newp_A$ ,  $p_B$ ,  $p_C$ , and  $p_D$ , which is  $LCM(6, 6, 4, 3) = 12$ .

If  $a_E$  is scheduled by the Multiple Processor A-h-k-a Scheduler using the information in the pre-run-time schedule including the processes  $newp_A$ ,  $p_B$ ,  $p_C$ ,  $p_D$  constructed by the Pre-run-time Scheduler above, then  $a_E$ 's worst-case response time will happen when  $a_E$  makes a request at time 3, and will be delayed by the conditions in Case 1 of the Multiple Processor A-h-k-a Scheduler at time 3.  $a_E$  will be executed from time 13 to time 15 after  $p_{B_1}$  has completed. The multiple processor simulation procedure for determining each A-h-k-a process' worst-case response time will simulate this execution in which  $a_E$ 's worst-case response time is  $RE(a_E, t_s) = RE(a_E, 3) = e'(a_E) - s'(a_E) = 15 - 3 = 12 \leq d_{a_E} = 14$ . In this case  $a_E$  is guaranteed to always meet its deadline. See Fig. 21.

Suppose the value chosen for  $threshold(a_A)$  is greater than 0.75, then A-h-k process  $a_A$  will not be converted into a new periodic process. If  $a_A$  is scheduled by the Multiple Processor A-h-k-a Scheduler using the information in the pre-run-time schedule including the processes,  $p_B$ ,  $p_C$ ,  $p_D$  constructed by the Pre-run-time Scheduler above, then  $a_A$ 's worst-case response time will happen when  $a_A$  makes a request at time 3, and will be delayed by the conditions in Case 1 of the Multiple Processor A-h-k-a Scheduler at time 3.  $a_A$  will be executed from time 8 to time 10 after  $p_{D_2}$  has completed.  $a_E$ 's response time is  $RE(a_A, t_s) = RE(a_A, 3) = e'(a_A) - s'(a_A) = 10 - 3 = 7 \leq d_{a_A} = 14$ . In this case  $a_A$  is guaranteed to always meet its deadline. See Fig. 22. However, while it is possible to guarantee that both  $a_A$  and  $a_E$  will always meet their respective deadlines when  $a_A$  is converted into a new periodic process  $newp_A$  as illustrated in Fig. 21; it is interesting to note that, if  $a_A$  is not converted into a new periodic process and remains asynchronous, then it would be impossible to guarantee that  $a_E$  will always meet its deadline, because there exist certain times, e.g., any time between 13 and 16, at which, if  $a_E$  is put into execution, it may be preempted by  $a_A$  and cause  $p_B$  to miss its deadline. Prohibiting  $a_A$  from preempting  $a_E$  by adding the exclusion relation  $a_E$  *excludes*  $a_A$  is not a solution, because not allowing  $a_A$  to preempt  $a_E$  will increase  $a_A$ 's worst-case response time to exceed  $a_A$ 's deadline. This example illustrates

that, in certain cases, it may be advantageous to convert an asynchronous process with a hard deadline and known characteristics into a new periodic process and schedule it before run-time.

5           It should be noted that various other embodiments of the present invention may be designed.

For example, tables of safe start time intervals for asynchronous processes may be used. In the methods described prior to the above description of scheduling  
10 processes on multiple processors, each periodic process was not restricted to execute within the time slot that was used to reserve processor capacity for that periodic process in the pre-run-time schedule. However, it is possible to enforce the requirement that every periodic  
15 process must execute strictly within its reserved time slot in the pre-run-time schedule, simply by changing each periodic process  $p$ 's release time  $R_p$  to be equal to the beginning of its time slot in the pre-run-time schedule, i.e., set  $R_p = s(p)$  for every  $p$ .

20           One advantage of doing this, is that it will make the execution times of periodic processes highly predictable, and thus allow construction of tables of "safe start time intervals" for asynchronous processes before run-time. Such tables would allow asynchronous  
25 processes to be scheduled at run-time by simple table lookup, and substantially reduce the run-time overhead of scheduling asynchronous processes.

The safe start time intervals can be computed by using the A-h-k-a Scheduler (or Multiple Processor A-h-  
30 k-a Scheduler) to determine, for each point in time  $t$  of the pre-run-time schedule, and for each processor  $k$ ,

whether each asynchronous process  $a_i$  can be put into execution at time  $t$  on processor  $k$ .

#### Example 15

5

For the asynchronous process  $a_E$ ,  $a_A$  in the system in Example 14, the following "safe start time tables" may be constructed.

When  $a_A$  is converted into a new periodic process  
10  $newp_A$ , and the pre-run-time schedule is as shown in Figure 21, the safe start time table for  $a_E$  should preferably contain the following "safe start time intervals":  
on processor 1: empty;  
15 on processor 2:  $\{(k * 12) + 1, (k * 12) + 2\}$ ,  $k = 0, 1, 2, \dots$

When  $a_A$  is not converted into a new periodic process, and the pre-run-time schedule is as shown in  
20 Figure 22, the safe start time table for  $a_A$  should preferably contain the following "safe start time intervals":  
on processor 1:  $\{(k * 6) + 2\}$ ,  $k = 0, 1, 2, \dots$   
on processor 2:  $\{(k * 12) + 2\}$ ,  $k = 0, 1, 2, \dots$

25

When  $a_A$  is not converted into a new periodic process, and the pre-run-time schedule is as shown in Figure 22, the safe start time table for  $a_E$  should preferably contain the following "safe start time  
30 intervals":  
on processor 1: empty



09336990-062199  
If some set of asynchronous processes is to be treated at lower priority than some other set of asynchronous processes, then they could be scheduled in a way similar to the way A-s-k processes and A-h-k processes have been scheduled as described earlier.

For example, although in this specification A-s-k processes are described as having been scheduled at a priority level that is lower than P-s-k processes, a different set of asynchronous processes with soft deadlines and known characteristics, say A-s-k-2, could have been chosen, that is scheduled at the same priority level as the P-s-k processes. Then the relationship between the A-s-k-2 processes and the P-s-k processes could be handled in a way that is similar to the way the relationship between the A-h-k and P-h-k processes was handled.

When using the integration method to schedule processes with different priority levels, the following general rules should be observed:

- (a) Periodic processes with known characteristics should be scheduled before run-time. The worst-case response times of asynchronous processes with known characteristics should also be determined before run-time. At run-time, the asynchronous processes should be scheduled in a way that guarantees that the timing constraints of all processes scheduled in the pre-run-time phase are always satisfied. Processes with unknown characteristics should be scheduled at a priority level that is lower than that of all processes with known characteristics.

(b) One should schedule the set of processes that have higher priority first and make sure that all their timing constraints are satisfied, before using the remaining processor capacity to schedule a set of  
5 processes that have lower priority.

In the present invention, different methods for scheduling each type of process, while observing the general rules above, may be used.

The present invention can also be used in an on-  
10 line mode. In such cases, in Step 2, when constructing a feasible pre-run-time schedule, when using the method described in the aforementioned 1990 article by Xu and Parnas, instead of constructing a search tree, the method can be used in a way such that it always only  
15 constructs one schedule for each set of processes, which should be as fast as any existing method, and still provide better chances of finding a feasible pre-run-time schedule than other methods.

20 As noted earlier, a simplified procedure for Step 4, pre-run-time scheduling of periodic processes with soft deadlines and known characteristics, will now be described.

In order to try to find a feasible schedule of  
25 all soft-deadline periodic processes together with all the guaranteed hard-deadline periodic processes using the optimal method, if a feasible schedule does not exist, then find an optimal schedule.

```

discontinue:= false;
nocandidate:= false;
while not discontinue do
begin{try to construct a feasible schedule}
  if  $e'(p_l) - D_{p_l} = \max\{e'(p_i) - D_{p_i} \mid p_i \in P\text{-h-k} \vee p_i \in P\text{-s-k}\}$  and  $e'(p_l) > D_{p_l}$ 
    { $p_l$  is the latest process and  $p_l$  is late}
  then
    begin{try to reduce the lateness of latest process  $p_l$ }
      candidatefound:= false;
      CandidateSet:= P-s-k
      while not candidatefound and not nocandidate do
        begin{search for candidate to revise deadline}
          select  $p_j$  such that
             $p_j \in \text{CandidateSet} \wedge d_{p_j} < d_{upplimit}(p_j) \wedge$ 
             $\text{criticality}(p_j) = \min\{\text{criticality}(p_i) \mid p_i \text{ in critical set } Z(p_l)\}$ 
            { $Z(p_l)$  is a set of processes that includes the latest
            process  $p_l$  and there is no time during which the processor
            is idle between their execution. (see [24])}
          if no such  $p_j$  exists then
            nocandidate:= true;
          else
            begin
              if  $e'(p_l) + 1 \leq d_{upplimit}(p_j)$ 
              then
                begin
                  if  $j \neq l$ 
                  then  $d_{p_j} := e'(p_l) + 1$ 
                    {this will cause  $p_j$  to
                    be scheduled after  $p_l$ 
                    which may help reduce lateness}
                  else  $d_{p_j} := e'(p_l)$ ;
                    {if  $p_j$  itself is latest
                    then set  $d_{p_j}$  such
                    that  $p_j$  will not be late}
                  candidatefound:= true
                end
              else remove  $p_j$  from CandidateSet
            end
        end
      end
    end
  end
end

```

```

        {scheduling  $p_j$  after  $p_l$ 
        is futile}

    end;
end{search for candidate to revise deadline};
if nocandidate
then
begin
    output  $Z(p_l)$ ;
    discontinue:= true;
    {the user must modify the
    characteristics of one or more
    processes in  $Z(p_l)$  before
    a feasible schedule can be found}

end
else
begin
    Use the optimal algorithm in [24] to try to find
    a feasible schedule again with the revised  $d_{p_j}$ ;
    if feasible does not exist, then find an optimal schedule;
end;
end{try to reduce the lateness of latest process  $p_l$ }
else discontinue:= true; {feasible schedule found}
end{try to construct a feasible schedule};
{Verify the schedulability of all A-h-k-a processes}
for each  $a_i \in \text{A-h-k-a}$  do
    if not discontinue
    then
    begin
        while  $RE_{a_i} > L_{a_i}$  and not discontinue do
        begin
            let CandidateSet:=  $\{ p_j \mid p_j \in \text{P-s-k} \wedge L_{p_j} \leq L_{a_i} \}$ 
            select  $p_j$  such that  $p_j \in \text{CandidateSet}$  and
            criticality( $p_j$ ) =  $\min\{\text{criticality}(p_i) \mid p_i \in \text{CandidateSet}$ 
             $\wedge d_i < d_{uplimit}(p_i)\}$ 
            if no such  $p_j$  exists then
                discontinue:= true;
                {the user must increase the deadline upper limit of some
                P-s-k process in CandidateSet}
            else
                 $d_{p_j} := d_{p_j} + 1$ ;
            end; {while}
        end; {for}
    end.

```



A method and a simulation procedure for determining the response times of the A-s-k processes, for example  $a_i$  follows:

For all  $a_i \in A-s-k$ :

$$RE_{a_i} = c_{a_i} + \text{DelayA}(a_i, RE_{a_i}) + \text{DelayP}(a_i, RE_{a_i}) + B(a_i)$$

where

$$\text{DelayA}(a_i, RE_{a_i}) = \sum_{a_j \in A-h-k \vee (a_j \in A-s-k \wedge L_{a_j} \leq L_{a_i} \wedge j \neq i)} \left\lceil \frac{RE_{a_i}}{\min_{a_j}} \right\rceil \cdot c_{a_j}$$

and

$$\text{DelayP}(a_i, RE_{a_i}) = \sum_{p_j \in P-h-k \vee p_j \in P-s-k} \left\lceil \frac{RE_{a_i}}{\text{prd}_{p_j}} \right\rceil \cdot c_{p_j}$$

and

$$B(a_i) = \max\{c_{a_j} \mid (a_j \in A-s-k \wedge L_{a_j} > L_{a_i} \wedge \exists x_k, x_k \in A-s-k: a_j \text{ excludes } x_k \wedge L_{x_k} \leq L_{a_i})\}$$

Note: In the above formula, the value of  $c_{p_j}$  is the original computation time of  $p_j$  (it does not include the time reserved for A-h-k-a processes with shorter deadlines).

The following procedure can be used to compute the worst-case response time of each A-s-k process:

```

i:= 0;
failure:= false;
while i ≤ number-of-A-s-k-processes and not(failure) do
begin
  if  $a_i \in A-s-k$ 
  then
  begin
     $RE_{new_i} := c_{a_i}$ ;
    responsetimefound:= false;
    while not(responsetimefound) and not(failure) do
    begin
       $RE_{previous_i} := RE_{new_i}$ ;
       $RE_{new_i} = \text{DelayA}(a_i, RE_{previous_i}) + \text{DelayP}(a_i, RE_{previous_i}) + B(a_i)$ ;
      if  $RE_{previous_i} = RE_{new_i}$ 
      then
      begin
         $RE_{a_i} := RE_{new_i}$ ;
        responsetimefound:= true;
      end
      if ( $RE_{new_i} > \text{responsetimelimit}$ )
      then failure:= true
    end;
  end;
  i:= i + 1;
end;

```

See Example 9 for concerning use of the above procedure.

In the following description, the second method is described, which uses simulation to determine the worst-case response time of an A-s-k process. This method uses the functions of the A-h-k-a Scheduler and the Main Run-Time Scheduler, which are described earlier.

A preferred method for computing the response time of an A-s-k process  $a_i$  corresponding to a feasible pre-run-time schedule of guaranteed periodic processes comprising an initial part of the pre-run-time schedule  $S_0(t_0)$ , in the interval  $\{0, t_0\}$ ; and a repeating part of the pre-run-time schedule  $S_{LCM}(prd_{LCM})$ , in the interval  $\{t_0, t_0 + prd_{LCM}\}$  is as follows:

fail:= false;

for  $t_s := 0$  to  $t_0 + prd_{LCM} - 1$  do

begin

Use the Main Run-Time Scheduler to schedule  $a_i$  under the following assumptions:

(1)  $a_i$  arrives in the system at time  $t_s$ .

(2) Let every A-h-k-a process  $a_j$  arrive at the following instants in time:  $R_{a_j}(k) = t_s + k * min_{a_j}$ ,  $k = 0, 1, 2, \dots, \lfloor \frac{d_{a_i}}{min_{a_j}} \rfloor$ , and be scheduled before  $a_i$ .

{ all A-h-k-a processes arrive at the same time as  $a_i$  at time  $t_s$ , and are put into execution before  $a_i$ . }

(3) Let every A-s-k process  $a_j$ , such that  $L_{a_j} \leq L_{a_i}$  arrive at the following instants in time:  $R_{a_j}(k) = t_s + k * min_{a_j}$ ,  $k = 0, 1, 2, \dots, \lfloor \frac{d_{a_i}}{min_{a_j}} \rfloor$ , and be scheduled before  $a_i$  whenever

$a_j$  and  $a_j$  have both arrived and  $a_i$  has not yet started.

{ all A-s-k processes whose latitudes are less than or equal to  $a_i$ 's latitude arrive at the same time as  $a_i$  at time  $t_s$ , and are put into execution before  $a_i$ . }

(4) Let A-s-k process  $a_{j_1}$ , such that  $c_{a_{j_1}} = \max\{c_{a_j} | (a_j \in \text{A-s-k} \wedge L_{a_j} > L_{a_i} \wedge \exists x_k, x_k \in \text{A-s-k} \vee x_k \in \text{P-g: } a_j \text{ excludes } x_k \wedge L_{x_k} \leq L_{a_i})\}$  arrive at the following instants in time:  $R_{a_{j_1}}(k) = t_s - 1 + k * \min_{a_{j_1}}$ ,  $k = 0, 1, 2, \dots, \lfloor \frac{d_{a_i}}{\min_{a_{j_1}}} \rfloor$   
 $\{a_{j_1}$  arrives 1 time unit before  $a_i$  at time  $t_s - 1$ , and,  
if it can be put into execution at that time, it will block  $a_i$ .)

(5) If  $\exists p_l, p_l \in P\text{-g} \wedge s(p_l) \leq t_s < e(p_l)$ , then assume that  $s'(p_l) = s(p_l) \wedge e'(p_l) = e(p_l)$ ; {i.e.,  $p_l$  started at the beginning of its time slot and will complete its computation at the end of its time slot in the pre-run-time schedule that was computed using adjusted computation times.}

(6) If the end of the current instance of the repeating part of the pre-run-time schedule is reached, continue at the beginning of the next instance of the repeating part of the pre-run-time schedule.

5           (7) If  $a_i$ 's computation is completed before its response time limit, then record the completion time of  $a_i$  as its response time  $RE(a_i, t_s)$ , and exit from the current iteration of the loop for this value of  $t_s$ , and start the next iteration for  $t_s := t_s + 1$ .

10           (8) If  $a_i$ 's response time limit is reached but  $a_i$  has not yet completed its computation, then set  $fail := true$  and exit from the procedure.

end;

if not fail then

15            $RE_{ai} := \max\{RE(a_i, t_s) \mid t_s = 0, 1, \dots, LCM - 1\};$

It should be noted that while the above-described methods can be carried out in a software programmed processor, and have been described with such a processor as an example, they, or one or more steps in the methods, can alternatively be carried out by hardware such as gate arrays or by other fixed or programmable structures, or combinations of these structures.

25           A processor can be any physical or virtual structure which can perform one or more functions. A processor can be a machine, a computer processor, a logic device, an organization, etc. A process can be any task or job that requires a finite amount of time to  
30 complete on a certain processor. Computation time is



the output result of the operation of the processes by the processor.

The memory preferably also has a portion 7 for storing a pre-run-time schedule, and a portion 9 for  
5 storing a run-time schedule for execution of the processes stored in the memory 3.

In operation, the processor receives input signals which demand that processes stored in the memory 3 (or which are received from other control or interrupt  
10 inputs, not shown) are executed. As described earlier in this specification, some of these processes can be periodic and some can be asynchronous. The processor, operating using an operating system stored in the memory 3, obtains the characteristics of the various processes  
15 from the memory, and creates or simulates a pre-run-time schedule, then creates or simulates a run-time schedule which uses the pre-run-time schedule, as described earlier in this specification. It then executes the run-time schedule as described herein, providing an  
20 output to the output device or system.

Figure 24A provides another example of a multiprocessor system on which the methods described herein can be carried out. The real-time system consists of a controlling system having two parts, a  
25 pre-run-time scheduling subsystem 10 and a run-time scheduling subsystem 11, and a controlled system 12. The controlled system 12 has data/control buses linking the various components. The controlled system 12 is comprised of a set of periodic processes p1, p2, .. pn,  
30 a set of asynchronous processes a1, a2, ... aj, and one or more processors 13a, 13b, ... 13m. Each processor 13a - 13m may have local memory and shared memory. On each processor 13a - 13m the processes, either periodic

or asynchronous, may be executed. Each process can be a task that requires a finite amount of time to complete on a processor that may have memory resources. Examples of such processes include : a service routines that  
5 periodically monitors conditions on a physical system and sends a signal when the conditions meet a certain standard or a task that responds to random user initiated requests. Any of the processes can be a complete subsystem unto itself comprising a processor  
10 and a memory that requires the use of some services offered by controlled system 12 in order to execute other periodic or asynchronous processes.

The pre-run-time scheduling subsystem 10 consists of one or more processors 14 that are used to  
15 perform pre-run-time scheduling. The pre-run-time scheduling subsystem scheduler (not shown) acquires information about the periodic processes such as constraints, release time, deadline, offset, precedence, and exclusion relations. The scheduler then uses this  
20 information to determine whether each asynchronous process should be converted into a new periodic process or not. After converting any suitable subset of asynchronous processes into new periodic processes but before run-time of the processes, the pre-run-time  
25 scheduler constructs a pre-run-time schedule for all the periodic processes which satisfies all the constraints and relations defined on the periodic processes. The scheduler then makes this schedule available to the run-time scheduling subsystem 11.

30 The run-time scheduling subsystem 11 consists of one or more processors 15 that are used to perform run-time scheduling of the periodic and asynchronous processes. The run-time scheduling subsystem 11 uses